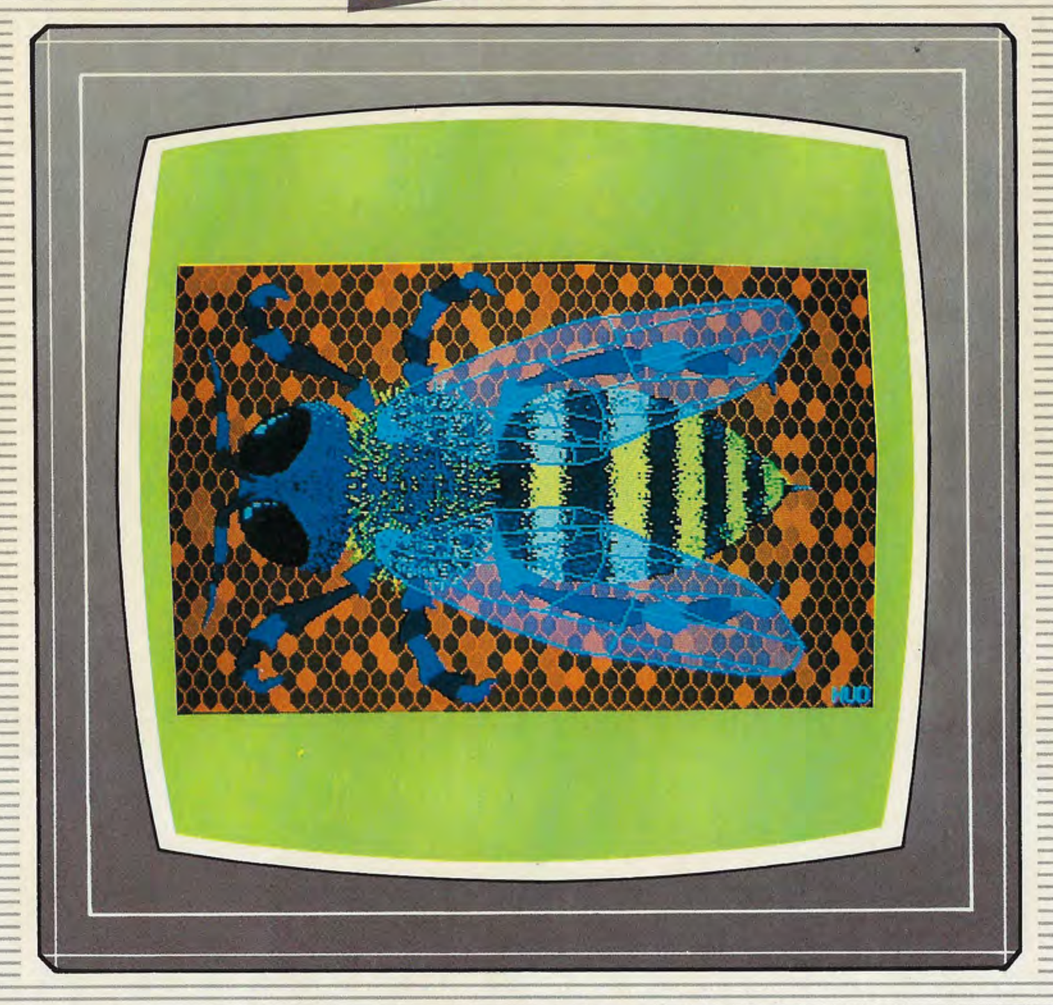
▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 9

PASO A PASO



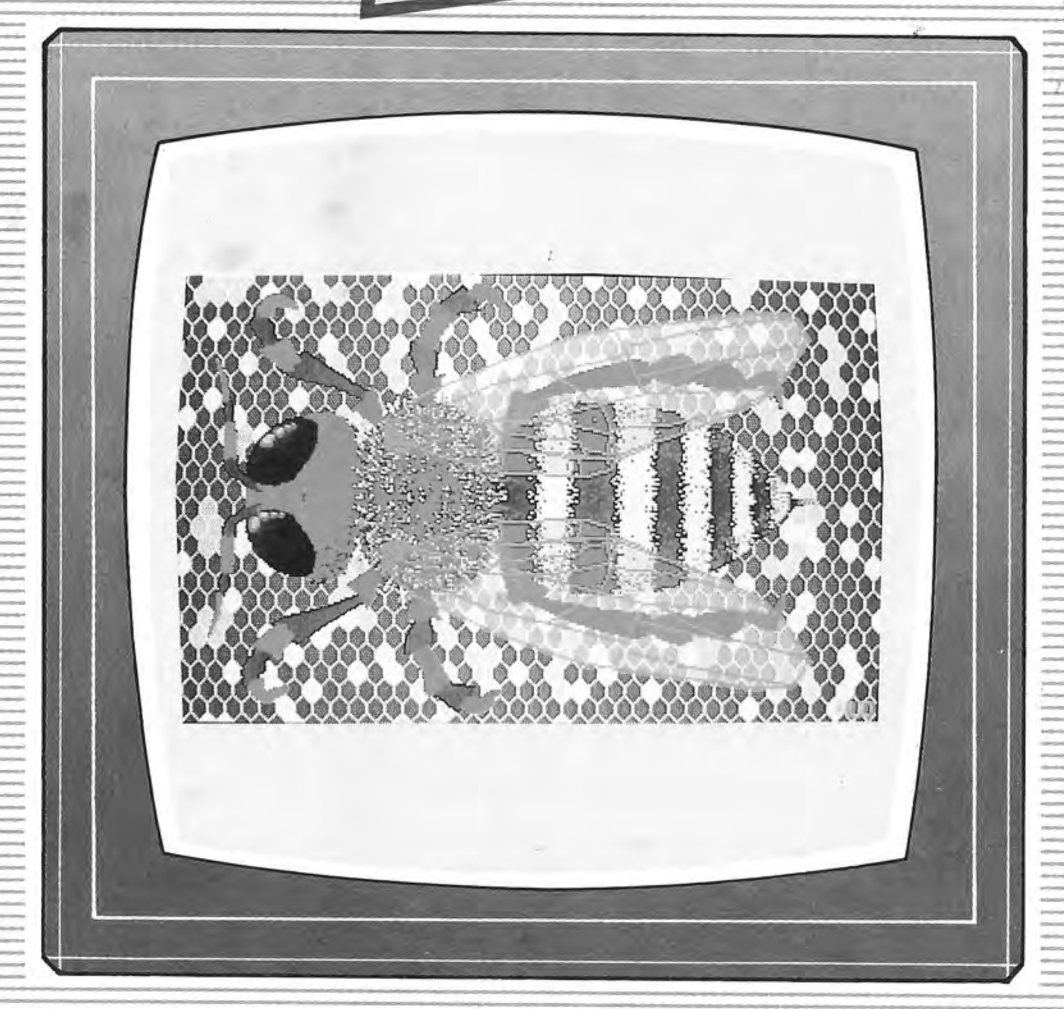
PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informatica y programacion 9

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomuniación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: Jesús Bocho, Licenciado en Informática. LOGO: Cristina Manzanero, Licenciada en Informática. APLICACIONES: Fernando Suero, Diplomado en Telecomunicación. OTROS LENGUAJES (Sistemas operativos): Domingo Villaseñor, Diplomado en Informática, y Lenguaje C: Enrique Serrano, Ingeniero en Telecomunicación.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-088-X.
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5.677-1987.
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Mayo, 1987.
P.V.P. Canarias: 335,–

INDICE

BASIC



Corrección de líneas de programa: EDIT

N muchas ocasiones puede suceder que cometamos algún error al teclear una línea de un programa. Si dicha línea no está almacenada aún en memoria, es decir, si no hemos pulsado INTRO, todavía estamos a tiempo de subsanar el error sobre la marcha. Todos los ordenadores disponen de dos flechas, una hacia la izquierda (←) y otra hacia la derecha (→), que permiten mover el cursor sobre la línea que estamos tecleando sin borrar nada. De este modo podemos situar el cursor donde esté el error. A continuación podemos hacer dos operaciones: borrar o insertar caracteres.

Para borrar caracteres los ordenadores disponen de una o varias teclas que pueden ser muy distintas según los fabricantes. En la tabla de la figura 1 podemos ver las teclas para borrar que podemos encontrar en los principales ordenadores, así como su efecto:

Amstrad	BORR	Borra el carácter situado a la izquierda del cursor.
	CLR	Borra el carácter situado debajo del cursor.
Commodore	DEL	Borra el carácter situado a la izquierda del cursor.
IBM	←	Borra el carácter situado a la izquierda del cursor.
	DEL	Borra el carácter situado debajo del cursor.
MSX	←	Borra el carácter situado a la izquierda del cursor.
	DEL	Borra el carácter situado debajo del cursor.
Spectrum	CAPS SHIFT + DELETE	Borra el carácter situado a la izquierda del cursor.

Si queremos insertar caracteres, en el AMSTRAD y en el SPECTRUM no tenemos más que teclear lo que deseemos y se irá insertando automáticamente a la izquierda de la posición en la que se encuentre el cursor. En el COMMODORE, el IBM y el MSX tendremos que pulsar la tecla INS antes de proceder a la inserción.

Por otra parte, puede suceder que detectemos el error después de haber introducido la línea en memoria. En este caso tendremos que utilizar el *editor* para corregir la línea. El formato general es el siguiente:

EDIT <número de línea>

donde *número de línea* es, evidentemente, el número de la línea del programa que deseamos corregir. Una vez tecleado este comando aparecerá en pantalla la línea deseada con el cursor sobre ella y, por tanto, podremos corregirla como ya sabemos. Al finalizar no tenemos más que pulsar INTRO.

Sin embargo, esta regla general no la cumplen ni el SPECTRUM ni el COMMODORE. En el caso del SPECTRUM podemos observar que siempre aparece, sobre la última línea introducida en memoria, un puntero (>) entre el número de línea y la instrucción. Si queremos corregir una línea tenemos que conseguir situar dicho puntero sobre ella. Para ello no tenemos más que pulsar simultáneamente CAPS SHIFT y la flecha para arriba (↑) o CAPS SHIFT y la flecha para abajo (↓) según deseemos que el puntero suba o baje. Una vez situado en la línea que queremos corregir pulsaremos simultáneamente CAPS SHIFT y EDIT y obtendremos la línea con el cursor encima, lista para ser corregida.

◄ *Formas de borrar caracteres en los diferentes ordenadores.*

En cuanto al COMMODORE, no dispone de la función EDIT y para corregir una línea podemos subir o bajar el cursor con las flechas ↑ o ↓ hasta situarlo en la línea deseada. A continuación podemos realizar las correcciones necesarias, como ya sabemos, y finalizar pulsando RETURN. Este método de corrección de líneas también es válido en el IBM (además de EDIT).

Finalmente hay que decir que si tecleamos dos líneas de programa con el mismo número de línea, en memoria sólo se almacenará la última que hayamos tecleado. Por tanto, si queremos sustituir una línea por otra nueva no tenemos más que teclear la nueva con el número de línea de la antigua. Si por el contrario, sólo deseamos suprimir una línea de la memoria no tenemos más que teclear el número de línea y pulsar INTRO.

La instrucción IF-THEN. Condiciones simples

Los ordenadores no sólo son capaces de ejecutar cálculos a gran velocidad, sino que además tienen capacidad para tomar *decisiones lógicas*.

Esta capacidad es posible gracias a la instrucción IF-THEN. Estas dos palabras BASIC actúan siempre juntas y tienen el siguiente formato:

IF <condición> THEN <lista de instrucciones> IF-THEN es equivalente a SI... ENTONCES... por tanto, la instrucción completa tiene un significado parecido a "Si se cumple la condición entonces ejecuta la lista de instrucciones". Pero veamos más detenidamente las *condiciones*.

Las condiciones pueden ser simples o compuestas, sin embargo, de momento vamos a centrarnos en la simples. Una condición simple tiene el siguiente formato:

<1.er miembro> Operador de relación <2.º miembro>

En la tabla de la figura 2 podemos ver todos los operadores de relación así como su significado.

Operador	Significado
=	Igual a
<>	Distinto de
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que


Tabla de operadores de relación.

En cuanto a los miembros que van a ambos lados del operador pueden ser datos constantes o variables de tipo numérico o alfanumérico, así como expresiones matemáticas.

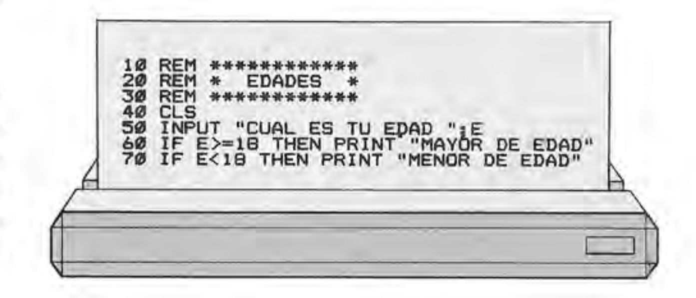
Veamos algunos ejemplos de condiciones:

```
NUM > = 0
A < B
N$ = "ANA"
P$ <> R$
F = 5 * A + B ↑ 2
```

En cualquier caso, ambos miembros de la condición deben ser del mismo tipo (numérico o alfanumérico).

En cuanto a la *lista de instrucciones* situada a continuación del THEN, se compone de una o varias instrucciones BASIC.

Visto esto ya podemos pasar a ver algunos ejemplos. El programa 1 sirve para determinar si el usuario es mayor de edad o no.



```
10 REM ************
20 REM *  EDADES  *
30 REM ************
40 CLS
50 INPUT "CUAL ES TU EDAD ";E
60 IF E>=18 THEN PRINT "MAYOR DE EDAD"
70 IF E<18 THEN PRINT "MENOR DE EDAD"
```

Las tres primeras líneas del programa sirven, como ya sabemos, para poner un título de una forma más elegante, al ir enmarcando entre asteriscos. La instrucción de la línea 40 borra la pantalla (en el COMMODORE sería 40 PRINT CHR$(147)). El INPUT de la línea 50 permite que el ordenador nos pregunte la edad; una vez tecleada la respuesta la ejecución continúa en la línea 60, donde el ordenador comprueba si la edad

tecleada (almacenada en la variable E) es mayor o igual que 18. Si se cumple esta condición entonces imprime en pantalla el mensaje MAYOR DE EDAD. A continuación pasa a la línea 70, donde comprueba si la edad es menor que 18. Evidentemente si se cumplió la condición anterior ésta no se cumplirá y, por tanto, no se ejecutará la instrucción situada tras el THEN. Si el caso fuese el contrario, es decir, la edad menor que 18, entonces no se verificaría la condición de la línea 60 y, por tanto, la ejecución del programa pasaría directamente a la línea 70 sin imprimir en pantalla MAYOR DE EDAD.

De aquí podemos sacar una conclusión importante: la lista de instrucciones que sigue al THEN sólo se ejecutará si se cumple la condición. En caso contrario, la ejecución pasará a la línea siguiente del programa, si la hay, y si no finalizará la ejecución.

Probemos ahora a sustituir la línea 70 del programa 1 por la siguiente:

```
70 PRINT "MENOR DE EDAD"
```

Si ahora ejecutamos el programa y probamos a responder, por ejemplo, 16 aparecerá en pantalla el mensaje MENOR DE EDAD, como era de esperar. Sin embargo, esto no significa que el programa funcione, ya que si lo volvemos a ejecutar, pero esta vez tecleando, por ejemplo, 24, en pantalla aparecerá el mensaje MAYOR DE EDAD y debajo MENOR DE EDAD, lo que significa que algo falla.

Evidentemente, si no se cumple la condición de la línea 60 es porque E es menor que 18 y, por tanto, no sería necesaria la segunda condición. Sin embargo, si se cumple la condición de la línea 60 habría que indicarle al ordenador que debe finalizar la ejecución tras imprimir MAYOR DE EDAD. Para ello disponemos de la instrucción END, de modo que el nuevo programa sería el siguiente:



```
10 REM ************
20 REM *  EDADES  *
30 REM ************
40 CLS
50 INPUT "CUAL ES TU EDAD ";E
60 IF E>=18 THEN PRINT "MAYOR DE EDAD":END
70 PRINT "MENOR DE EDAD"
```

Podemos observar que en la línea 60 aparecen dos puntos (:) entre la instrucción PRINT y la instrucción END. Estos dos puntos permiten introducir más de una instrucción en una línea de programa. Por tanto, podemos teclear varias instrucciones en una sola línea siempre que estén separadas con dos puntos.

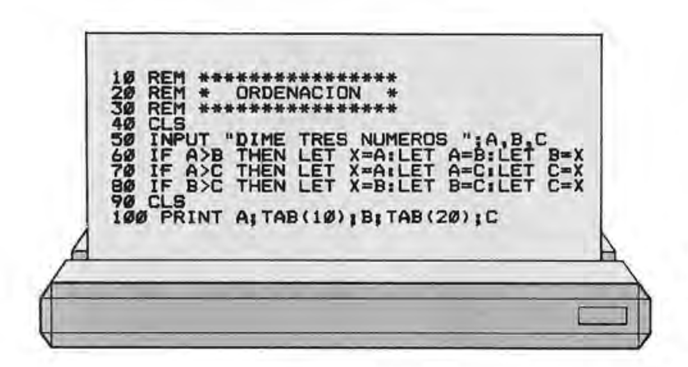
La instrucción END hace que la ejecución del programa se detenga en la línea 60 si se cumple la condición. En caso contrario, la ejecución continuará normalmente por la línea 80.

La instrucción END se encuentra en todos los ordenadores, excepto en el SPECTRUM. Sin embargo, esta falta se puede suplir con otra instrucción BASIC que actúa de forma similar: STOP. La diferencia entre END y STOP es que esta última permite reanudar la ejecución donde se había interrumpido mediante el comando CONTINUE.

De forma similar a como el ordenador compara números, puede comparar cadenas de caracteres. Podemos comprobarlo tecleando el programa 3.

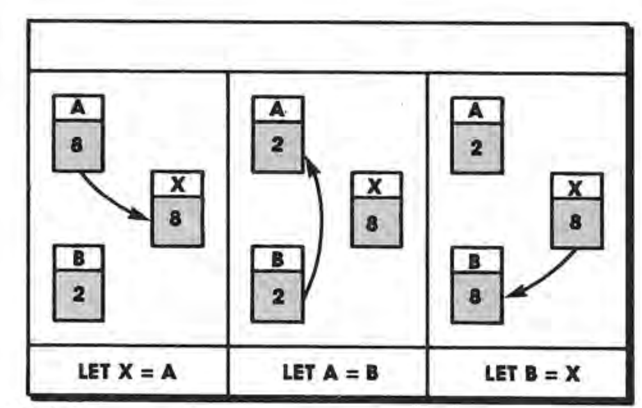
```
10 REM *************
20 REM *  NOMBRES  *
30 REM *************
40 CLS
50 LET N$="RAMON"
60 INPUT "COMO TE LLAMAS ";R$
70 PRINT
80 IF R$<>N$ THEN PRINT "HOLA ";R$:END
90 PRINT "YA CONOZCO OTRO ";R$
```

Por último, el programa 4 pide tres números y los ordena de menor a mayor, en base a compararlos dos a dos e intercambiarlos cuando proceda.



Con este programa queda claro el uso de los dos puntos para construir líneas multisentencia (con varias instrucciones).

Sólo si se cumple la condición se realiza el intercambio del contenido de las variables, ayudándose de una variable intermedia X, tal y como podemos ver en el ejemplo esquematizado de la figura.



Esquema de intercambio del contenido de las variables A y B utilizando la variable intermedia X.

Del mismo modo que la máquina ordena números, puede ordenar cadenas de caracteres alfabéticamente. No tenemos más que introducir los siguientes cambios en el programa 4:

— Cambiar A, B, C y X por A$, B$, C$ y X$, respectivamente.

— En la línea 50 cambiar el mensaje por DIME TRES NOMBRES y ahora ejecutemos el programa.



MAQUINA Z-80

OPERACIONES CON 8 BITS Y BCD

EN esta parte trataremos las operaciones aritméticas y lógicas que el Z-80 puede realizar sobre 8 bits, en código natural o en BCD (Decimal Codificado en Binario).

Ante todo debemos aclarar que la parte del Z-80 encargada de realizar las operaciones aritmético-lógicas (ULA) puede operar simultáneamente sobre 16 bits, que es una característica propia de un microprocesador de 16 bits. Por ser el bus externo de 8 bits, el uso de operandos de 16 bits necesitará algunos ciclos adicionales para realizar las operaciones; debido a esto se considera al Z-80 como un microprocesador de 8 bits.

Suma de dos números

La principal instrucción es ADD (sumar en inglés). Esta suma dos números de 8 bits, siendo uno de éstos el acumulador. La suma de dos números de 8 bits puede generar números de 9 bits.

```
   10101010
 + 10101010
 ----------
 1 01010100
```

Acarreo

El defecto es que tenemos que cargar previamente el contenido de la dirección en el registro HL.

— Además, existen los direccionamientos indirectos preindexados con los registros IX e IY.

ADD A,(IX+d)
ADD A,(IY+d)

La otra instrucción existente para realizar sumas es la ADC (ADd with Carry, sumar con acarreo). Esta instrucción es igual que la ADD, sólo que además se sumaría el acarreo que pudiera existir de anteriores operaciones.

De esta forma se pueden sumar números de más de 8 bits con instrucciones de 8 bits.

```
   1010101010101010
 + 1010101010101010
 ------------------
 1 0101010101010100

           10101010
         + 10101010
         ----------
         1 01010100

   10101010
 + 10101010
 ----------
 1 01010100

 10101010101010100
```

Ejemplo de suma de 16 bits utilizando dos sumas de 8 bits con acarreo.

Los tipos de direccionamientos son los mismos que se vieron para ADD.

Existen instrucciones para incrementar en una unidad un determinado registro, o una posición de memoria. Si quisiéramos realizar esto usando la instrucción ADD serían necesarias tres instrucciones: cargar el dato en el acumulador, incrementarlo y mover el acumulador con el

resultado a la posición de la que se tomaron los datos. Para resumir esto se utilizará la instrucción INC.

Esta instrucción puede usarse con tres tipos de direccionamiento:

— Directo en un registro.
— Indirecto con el registro HL.
— Indirecto preincrementado con los registros IX e IY.



Resta de dos números

Las instrucciones dedicadas a este cometido son las SUB y SBC.

La instrucción SUB resta dos números de 8 bits, siendo uno de ellos el acumulador.

La instrucción SBC resta dos números, al igual que SUB, pero resta también el acarreo negativo (Borrow Carry). Hay que tener en cuenta que este acarreo es negado, y, por tanto, que si se quiere restar 1, no debe ponerse la bandera del carry, CY, a 1, sino a 0.

Los tipos de direccionamiento que se pueden usar son los mismos que se usaron para ADD y ADC.

Para completar la analogía con las operaciones de sumar aquí también existe una instrucción de decremento DEC que funciona igual que la INC, vista anteriormente, sólo que en vez de sumar una unidad, resta una unidad.

Existe una operación de resta que no almacena el resultado en él acumulado: CP. Esta sirve para comparar. Al realizar se puede afectar a las banderas que indican si el resultado es cero (Z) y si el resultado es positivo o negativo (N). Esto puede servir para realizar comparaciones ejecutando instrucciones de salto condicional, que se verán más adelante.

Al restar puede ser necesario cambiar el signo del resultado; para ello existen dos instrucciones:

— CPL: Complementa a uno el acumulador.
— NEG: Complementa a dos el acumulador.

Por último, existen dos instrucciones para controlar el estado de la bandera de acarreo CY.

— CCF: Complementa el indicador de arrastre, es decir, cambia de estado.
— SCF: Pone a 1 el indicador de arrastre.

Grupo aritmético y lógico de 8 bits

Código mnemotécnico	Operación simbólica	Indicadores S Z H P/V N C	Códigos 76 543 210 Hex	N.º de bytes	N.º de ciclos M	N.º de estados T	Comentarios
ADD A,r	A←A+r	↕ ↕ X ↕ X V 0 ↕	10 [000] r	1	1	4	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A,n	A←A+n	↕ ↕ X ↕ X V 0 ↕	11 [000] 110 ←n→	2	2	7	
ADD A,(HL)	A←A+(HL)	↕ ↕ X ↕ X V 0 ↕	10 [000] 110	1	2	7	
ADD A,(IX+d)	A←A+(IX+d)	↕ ↕ X ↕ X V 0 ↕	11 011 101 DD 10 [000] 110 ←d→	3	5	19	
ADD A,(IY+d)	A←A+(IY+d)	↕ ↕ X ↕ X V 0 ↕	11 111 101 FD 10 [000] 110 ←d→	3	5	19	
ADC A,s	A←A+s+CY	↕ ↕ X ↕ X V 0 ↕	[001]				s puede ser un r, n, (HL)m (IX+d), (IY+d) como para ADD; los códigos se forman como en ADD, pero reemplazando el [000] de ADD con los bits que se indican
SUB s	A←A−s	↕ ↕ X ↕ X V 1 ↕	[010]				
SBC A,s	A←A−s−CY	↕ ↕ X ↕ X V 1 ↕	[011]				
AND s	A←A∧s	↕ ↕ X 1 X P 0 0	[100]				
OR s	A←A∨s	↕ ↕ X 0 X P 0 0	[110]				
XOR s	A←A⊕s	↕ ↕ X 0 X P 0 0	[101]				
CP s	A−s	↕ ↕ X ↕ X V 1 ↕	[111]				
INC r	r←r+1	↕ ↕ X ↕ X V 0 •	00 r [100]	1	1	4	
INC (HL)	(HL)←(HL)+1	↕ ↕ X ↕ X V 0 •	00 110 [100]	1	3	11	
INC (IX+d)	(IX+d)←(IX+d)+1	↕ ↕ X ↕ X V 0 •	11 011 101 DD 00 110 [100] ←d→	3	6	23	
INC (IY+d)	(IY+d)←(IY+d)+1	↕ ↕ X ↕ X V 0 •	11 111 101 FD 00 110 [100] ←d→	3	6	23	
DEC m	m←m−1	↕ ↕ X ↕ X V 1 •	[101]				m es un r, (HL), (IX+d), (IY+d) como para INC; DEC tiene igual formato y estados que INC; el código se forma como en INC, pero reemplazando el [100] de INC por [101]

Grupo aritmético y de control de aplicación general

Código mnemotécnico	Operación simbólica	Indicadores S Z H P/V N C	Códigos 76 543 210 Hex	N.º de bytes	N.º de ciclos M	N.º de estados T	Comentarios
DAA	Ajusta el acumulador para operaciones de suma y resta en BCD(1) con operandos BCD	↕ ↕ X ↕ X P • ↕	00 100 111 27	1	1	4	Ajuste decimal del acumulador
CPL	$A \leftarrow \overline{A}$	• • X 1 X • 1 •	00 101 111 2F	1	1	4	Complementa (a 1) el acumulador
NEG	$A \leftarrow 0 - A$	↕ ↕ X ↕ X V 1 ↕	11 101 101 ED 01 000 100 44	2	2	8	Cambia de signo del acumulador (complemento a 2)
CCF	$CY \leftarrow \overline{CY}$	• • X X X • 0 ↕	00 111 111 3F	1	1	4	Complementa el indicador de arrastre
SCF	$CY \leftarrow 1$	• • X 0 X • 0 1	00 110 111 37	1	1	4	Pone a 1 el indicador de arrastre
NOP	No operación	• • X • X • • •	00 000 000 00	1	1	4	
HALT	Detiene el Z80	• • X • X • • •	01 110 110 76	1	1	4	
DI*	$IFF \leftarrow 0$	• • X • X • • •	11 110 011 F3	1	1	4	
EI*	$IFF \leftarrow 1$	• • X • X • • •	11 111 011 FB	1	1	4	
IM 0	Selecciona modo 0 para las interrupciones	• • X • X • • •	11 101 101 ED 01 000 110 46	2	2	8	
IM 1	Selecciona modo 1 para las interrupciones	• • X • X • • •	11 101 101 ED 01 010 110 56	2	2	8	
IM 2	Selecciona modo 2 para las interrupciones	• • X • X • • •	11 101 101 ED 01 011 110 5E	2	2	8	

NOTAS: (1) BCD: decimal codificado en binario.
IFF representa la báscula de habilitación de interrupciones.
CY representa el indicador de arrastre.
* indica que las interrupciones no son examinadas al final de DI o EI.

Operaciones lógicas

Las instrucciones de operaciones son:

— AND: Realiza la operación "Y" lógica entre los bits de los dos operandos.

— OR: Realiza la operación "O" lógica entre los operandos.

— XOR: Realiza la operación "O exclusivo" entre los dos operandos.

Los tipos de direccionamiento son los mismos que para las operaciones de suma.

Por último, hay que indicar que para realizar las operaciones en BCD es necesario ejecutar antes la instrucción DAA para activar el modo decimal.



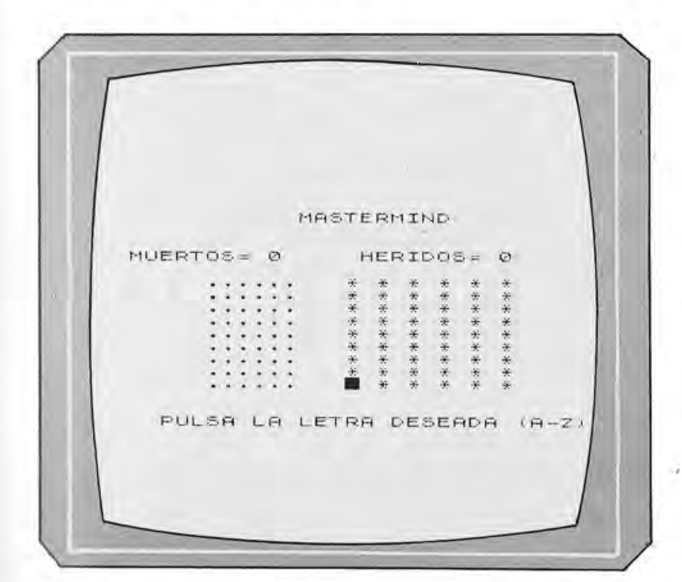
PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Programa: Mastermind para Spectrum

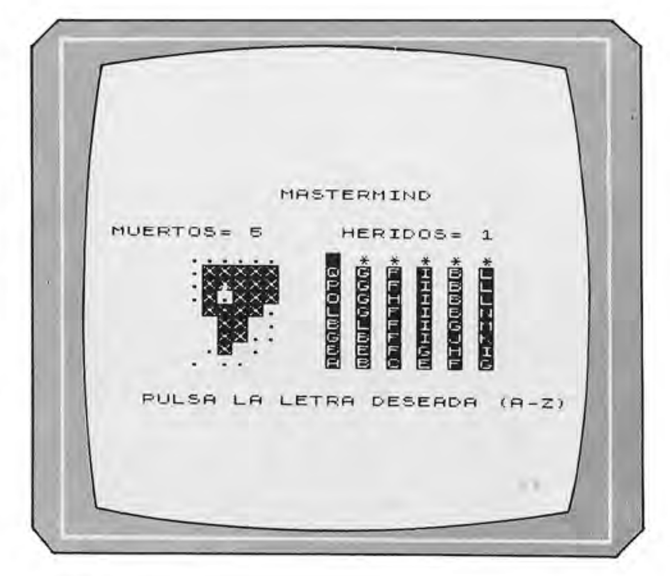
ESTE primer programa de este tomo nos va a permitir jugar al famosísimo juego del MASTERMIND. Lo más peculiar de este juego es que no es el típico MASTERMIND de números, sino que es de letras. Esto lo hace más difícil. Aparte de esta dificultad, nos encontramos con que las letras pueden repetirse, lo cual lo hace casi imposible. Esto es mejor de esta manera, ya que así se consigue que el juego no se vuelva aburrido, pues es necesario pensar más y mejor para llegar al final.



Comienzo de una partida de Mastermind.

Para aquellas personas que no sepan en qué consiste el juego, damos a continuación una breve explicación de las normas.

El juego consiste en adivinar una serie de seis letras que ha pensado el ordenador. La ordenación de estas letras es aleatoria, esto es, no tienen por qué formar ninguna palabra. Si la forman es por casualidad.

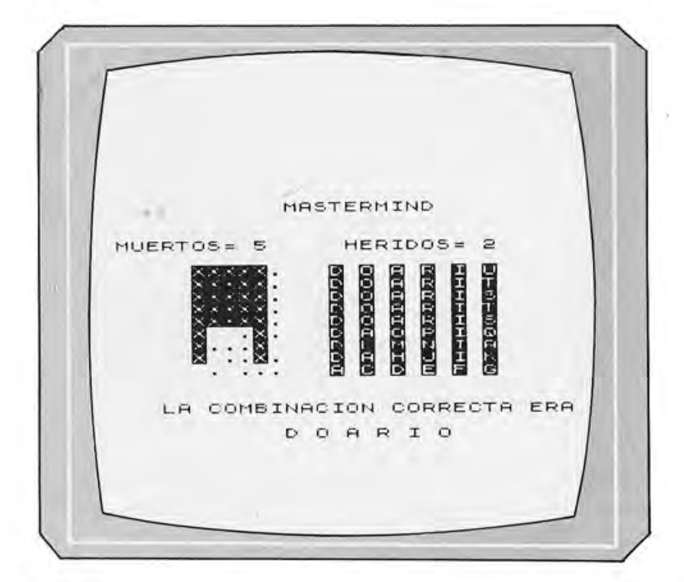




A punto de acabar la partida.

Para adivinar la serie de letras que ha elegido el ordenador, vamos proponiendo nuestra propia serie de letras. Cada vez que terminemos una serie el ordenador nos dirá el número de muertos y de heridos que tenemos. Pero ¿qué significa un muerto y un herido? Cada vez que acertemos una letra y ésta esté colocada en la misma posición en la que la tiene el ordenador, entonces dicha letra

será un muerto. Si hemos acertado la letra pero no la posición, entonces es un herido. Hay que decir que hay que adivinar no sólo la serie de letras, sino también la ordenación que tienen dichas letras.



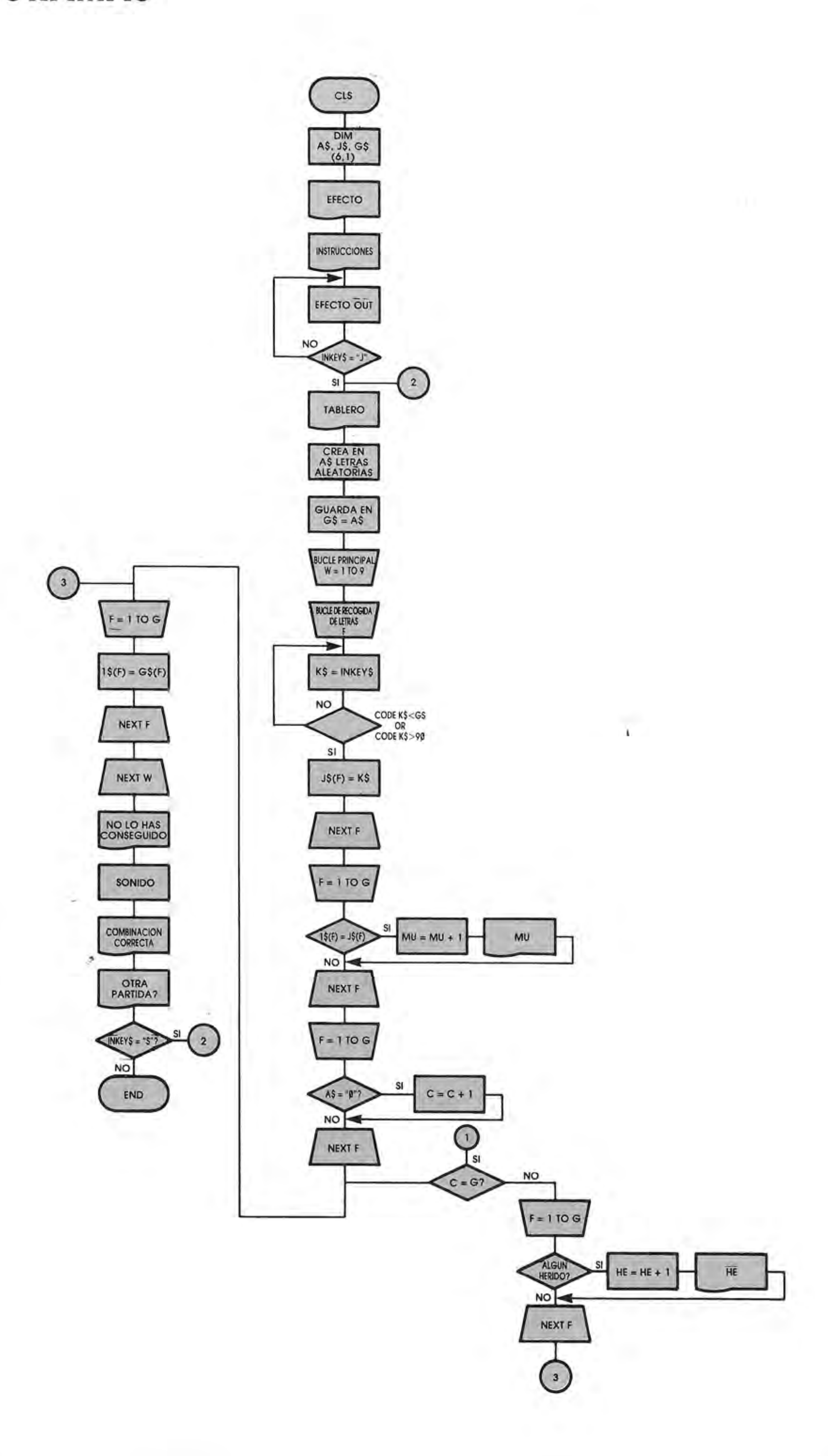
Cuando agotamos nuestras posibilidades, si no hemos acertado, el ordenador nos dice la serie que había pensado.

El jugador sólo dispone de 9 oportunidades para adivinar la secuencia.

```
 10 REM **********************
 20 REM ***** MASTERMIND ******
 30 REM **********************
 40 REM ******** POR **********
 50 REM **********************
 60 REM **** CARLOS DORAL *****
 70 REM **********************
 80 BORDER 7
 90 PAPER 7
100 INK 0
110 CLS
120 DIM a$(6,1)
130 DIM g$(6,1)
140 DIM j$(6,1)
150 LET l=10
160 LET b$="MASTERMIND"
170 FOR c=1 TO L
180   FOR f=0 TO 9+1
190     PRINT AT 10,f;" "; PAPER
2; INK 6;b$(1 TO 1)
200   NEXT f
210   LET l=l-1
220 NEXT c
230 PRINT AT 12,0;"  Pulsa las
teclas A-Z para se- leccionar tu
s 6 letras y consi- gue adivinar
 las de la maquina."
240 OUT 254,INT (RND*255)
250 PRINT AT 21,5;"PULSA SPACE
PARA JUGAR"
260 IF INKEY$<>" " THEN  GO TO
240
270 OUT 254,7
280 CLS
290 RANDOMIZE PEEK 23672
300 PRINT AT 0,11; FLASH 1; INK
 7; PAPER 1; BRIGHT 1;"MASTERMIN
D"
310 PRINT AT 3,0;"MUERTOS= 0";"
    HERIDOS= 0"
320 FOR f=1 TO 6
330   LET a$(f)=CHR$ (65+INT (R
ND*25))
340 NEXT f
350 FOR f=1 TO 6
360   LET g$(f)=a$(f)
370 NEXT f
380 FOR f=5 TO 13
390   PRINT AT f,4;" ......    *
 * * * * *"
400 NEXT f
410 LET x=14
420 LET y=13
430 FOR w=1 TO 9
440   LET he=0
450   LET mu=0
460   FOR f=1 TO 6
470     PRINT AT y,x; INVERSE 1;
" "
480     PRINT AT 16,2;"PULSA LA
LETRA DESEADA (A-Z)"
490     PRINT AT 3,9; OVER 1;" "
;AT 3,24;" "
500     POKE 23658,8
510     LET k$=INKEY$
520     IF k$="" THEN  GO TO 500
530     IF CODE k$<65 OR CODE k$
>90 THEN  GO TO 500
540     PRINT AT y,x; INVERSE 1;
k$
550     LET j$(f)=k$
560     LET x=x+2
570     FOR p=1 TO 50
580     NEXT p
590   NEXT f
600   FOR f=1 TO 6
610     IF a$(f)=j$(f) THEN  LET
 a$(f)="0": LET j$(f)="0": LET m
u=mu+1: PRINT AT 3,9; FLASH 1;mu
: PRINT AT y,f+4; INVERSE 1;"X"
620   NEXT f
630   LET c=0
640   FOR f=1 TO 6
650     IF a$(f)="0" THEN  LET c
=c+1
660   NEXT f
670   IF c=6 THEN  GO TO 990
680   LET c=1
690   FOR f=1 TO 6
700     IF j$(c)<>"0" AND a$(f)<
>"0" AND j$(c)=a$(f) THEN  LET j
$(c)="0": LET a$(f)="0": LET he=
he+1: PRINT AT y,c+4; FLASH 1;"
";AT 3,24;he
710   NEXT f
720   LET c=c+1
```

```
730   IF c<>7 THEN  GO TO 690
740   LET y=y-1
750   LET x=14
760   LET c=0
770   FOR f=1 TO 6
780    LET a$(f)=g$(f)
790   NEXT f
800  NEXT w
810  PRINT AT 16,0; INVERSE 1;"
NO LO HAS CONSEGUIDO,LO SIENTO
"
820  FOR F=30 TO 0 STEP -3
830    BEEP .20,F
840  NEXT F
850  PRINT AT 16,0; FLASH 1;"
LA COMBINACION CORRECTA ERA  "
860  LET c=1
870  FOR f=11 TO 21 STEP 2
880   IF a$(c)="0" THEN  PRINT
AT 18,f;"X"
890   IF a$(c)<>"0" THEN  PRINT
AT 18,f;a$(c)
900   LET c=c+1
910  NEXT f
920 FOR F=1 TO 1000
930 NEXT F
940 PRINT AT 18,11;"
"
950 PRINT AT 16,0;"QUIERES JUGA
R OTRA PARTIDA("; FLASH 1;"S"; F
LASH 0;"/"; FLASH 1; INVERSE 1;"
N"; FLASH 0; INVERSE 0;")?"
960 IF INKEY$="S" OR INKEY$="s"
THEN  GO TO 280
970 IF INKEY$="N" OR INKEY$="n"
THEN  GO TO 1160
980 GO TO 960
990 REM ***********************
1000 REM *** FINAL DEL JUEGO ***
1010 REM ***********************
1020 LET c=1
1030 FOR f=14 TO 24 STEP 2
1040   PRINT AT y,f; BRIGHT 1; P
APER 7; INK 1;g$(c)
1050   LET c=c+1
1060 NEXT f
1070 PRINT AT 16,0; FLASH 1;"
       LO CONSEGUISTE        "
1080 FOR F=0 TO 30 STEP .3
1090   BEEP .001,f
1100 NEXT f
1110 PRINT AT 19,2;"QUIERES OTRA
PARTIDA (S/N)?"
1120 BEEP .02,INT (RND*50)
1130 IF INKEY$="N" OR INKEY$="n"
THEN  STOP
1140 IF INKEY$="S" OR INKEY$="s"
THEN  GO TO 280
1150 GO TO 1120
```

A continuación se da el organigrama del programa para que podáis entender mejor cómo funciona e incluso modificarlo para que sea más fácil (o más difícil), para que dé más oportunidades.

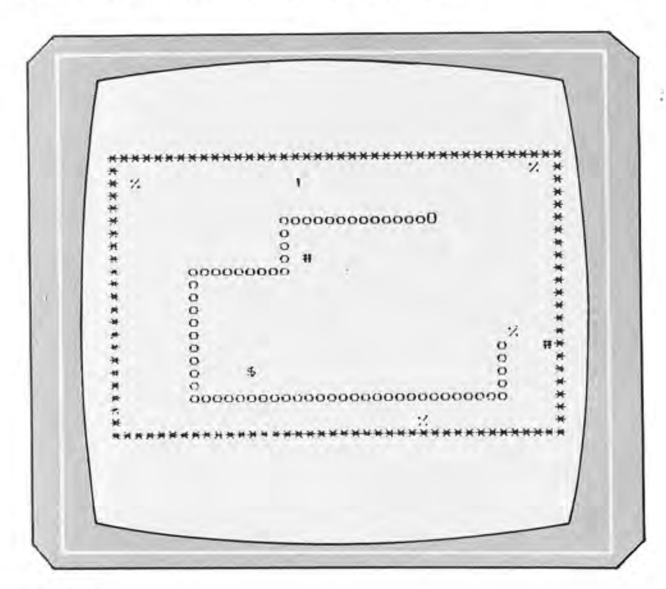

CLS
DIM A$, J$, G$ (6,1)
EFECTO
INSTRUCCIONES
EFECTO OUT
NO
INKEY$ = "J"
SI
2
TABLERO
CREA EN A$ LETRAS ALEATORIAS
GUARDA EN G$ = A$
BUCLE PRINCIPAL W = 1 TO 9
BUCLE DE RECOGIDA DE LETRAS F
K$ = INKEY$
NO
CODE K$<GS OR CODE K$>9Ø
SI
J$(F) = K$
NEXT F
F = 1 TO G
1$(F) = J$(F)
SI
MU = MU + 1
MU
NO
NEXT F
F = 1 TO G
A$ = "Ø"?
SI
C = C + 1
NO
NEXT F
1
SI
C = G?
NO
F = 1 TO G
ALGUN HERIDO?
SI
HE = HE + 1
HE
NO
NEXT F
3
3
F = 1 TO G
1$(F) = G$(F)
NEXT F
NEXT W
NO LO HAS CONSEGUIDO
SONIDO
COMBINACION CORRECTA
OTRA PARTIDA?
INKEY$ = "S"?
SI
2
NO
END

Programa: Gusano loco

El último programa de este fascículo es un juego. En él tenemos que intentar que un gusano que se mueve por la pantalla se coma el máximo número de frutas que pueda. El programa aparece cuando, después de comerse una fruta, el gusano aumenta su longitud. Las teclas que tienes que utilizar para mover el gusano por la pantalla son:

O - Izquierda	Q - Arriba
P - Derecha	A - Abajo

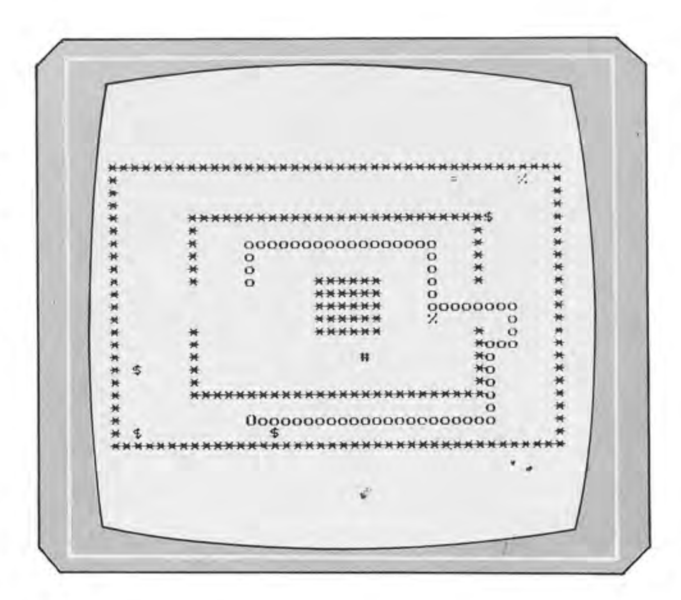
Tienes que tener cuidado con no chocarte con las paredes ni contigo mismo, pues en ese caso perderías una de las tres vidas de las que dispones.





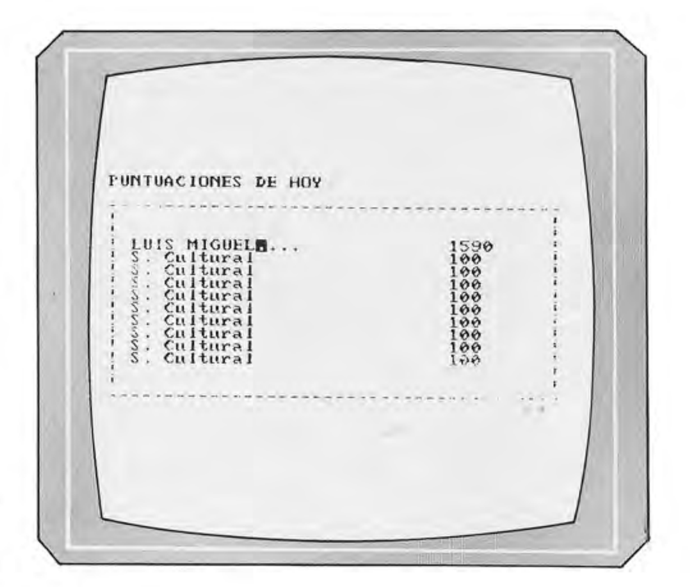
Pantalla del juego «Gusano loco».

El programa cuenta con cinco niveles distintos de dificultad entre 0 y 4 (0 = muy rápido. 4 = muy lento). Aparte de estos cuatro niveles, dentro del programa hay definidas cuatro pantallas distintas. El jugador pasa de una a otra cuando se ha comido 15 frutas en la pantalla en la que está actualmente. Cada pantalla es más difícil que la anterior.





Pantalla del juego «Gusano loco».



Al final del juego podemos introducir nuestro nombre si tenemos una puntuación alta.

```
100 REM **************************
110 REM * JUEGO DEL GUSANO LOCO *
111 REM *                        *
112 REM * POR Fco. Morales       *
113 REM **************************
```

```
114 REM
115 REM *************************
116 REM * (c) Ed. Siglo Cultural *
117 REM * (c) 1987               *
120 REM *************************
130 REM
140 REM ****************************
150 REM * PANTALLA DE PRESENTACION *
160 REM ****************************
170 REM
180 CLS
190 PRINT "GUSANO LOCO"
200 PRINT "-----------"
210 PRINT:PRINT:PRINT
220 PRINT "USA LAS SIGUIETES TECLAS:"
230 PRINT:PRINT
240 PRINT "O - IZQUIERDA"
250 PRINT "P - DERECHA"
260 PRINT "Q - ARRIBA"
270 PRINT "A - ABAJO"
280 PRINT:PRINT
290 INPUT "QUE NIVEL QUIERES (0-4) ";NN
300 IF NN<0 OR NN>4 THEN GOTO 290
310 LET NN=25*NN
320 REM
330 REM ******************************
340 REM * INICIALIZACION DEL PROGRAMA *
350 REM ******************************
360 REM
370 CLS
380 PRINT "ESPERA UN MOMENTO"
390 DIM A$(10):DIM S(10):DIM F$(4)
400 FOR I=1 TO 10
410    LET A$(I)="S. Cultural"
420    LET S(I)=100
430 NEXT I
440 LET NV=1:LET VV=3
450 DIM A(40,23)
460 LET F$(1)="#"
470 LET F$(2)="%"
480 LET F$(3)="$"
490 LET F$(4)="="
500 REM
510 REM ********************
520 REM * PROGRAMA GENERAL *
530 REM ********************
540 REM
550 FOR I=1 TO 40
560    FOR J=1 TO 23
570       LET A(I,J)=0
580    NEXT J
590 NEXT I
600 LO=0
610 CLS
620 LET X=10:LET Y=10
630 LET X1=10:LET Y1=10
640 LET X3=10:LET Y3=10
650 GOSUB 710
660 ON NV GOSUB 900,960,1100,1360
670 GOSUB 1720
680 GOSUB 2040
690 FOR J=1 TO NN:NEXT J
700 GOTO 670
710 REM
720 REM ***************************************
730 REM * DIBUJO DE LOS LIMITES DE LA PANTALLA *
740 REM ***************************************
750 REM
760 FOR I=1 TO 40
```

```
770     PRINT "*";
780     LET A(I,1)=1
790 NEXT I
800 FOR I=2 TO 22
810     PRINT "*";TAB(40);"*"
820     LET A(1,I)=1
830     LET A(40,I)=1
840 NEXT I
850 FOR I=1 TO 40
860     PRINT "*";
870     LET A(I,23)=1
880 NEXT I
890 RETURN
900 REM
910 REM ******************
920 REM * PANTALLA No. 1 *
930 REM ******************
940 REM
950 RETURN
960 REM
970 REM ******************
980 REM * PANTALLA No. 2 *
990 REM ******************
1000 REM
1010 FOR I=5 TO 19
1020     LOCATE I,8
1030     PRINT "*"
1040     LET A(8,I)=1
1050     LOCATE I,33
1060     PRINT "*"
1070     LET A(33,I)=1
1080 NEXT I
1090 RETURN
1100 REM
1110 REM ******************
1120 REM * PANTALLA No. 3 *
1130 REM ******************
1140 REM
1150 FOR I=8 TO 33
1160     LOCATE 5,I
1170     PRINT "*"
1180     LET A(I,5)=1
1190     LOCATE 19,I
1200     PRINT "*"
1210     LET A(I,19)=1
1220 NEXT I
1230 FOR I=6 TO 18
1240     LOCATE I,8
1250     PRINT "*";TAB(33);"*"
1260     LET A(8,I)=1
1270     LET A(33,I)=1
1280 NEXT I
1290 FOR I=11 TO 13
1300     LOCATE I,8
1310     PRINT " ";TAB(33);" "
1320     LET A(8,I)=0
1330     LET A(33,I)=0
1340 NEXT I
1350 RETURN
1360 REM
1370 REM ******************
1380 REM * PANTALLA No. 4 *
1390 REM ******************
1400 REM
1410 GOSUB 1100
1420 FOR I=10 TO 14
1430     LOCATE I,19
1440     PRINT "******"
1450     FOR J=19 TO 24
```

```
1460       LET A(J,I)=1
1470    NEXT J
1480 NEXT I
1490 RETURN
1500 REM
1510 REM **************************
1520 REM * CHOQUE CONTRA LA PARED *
1530 REM **************************
1540 REM
1550 CLS
1560 PRINT "TE HAS CHOCADO CONTRA LA PARED."
1570 PRINT "LA PROXIMA VEZ HAS DE TENER MAS CUIDADO"
1580 IF VV=1 THEN GOTO 2450
1590 LET VV=VV-1
1600 GOTO 550
1610 REM
1620 REM ***************************
1630 REM * CHOQUE CONTRA UNO MISMO *
1640 REM ***************************
1650 REM
1660 CLS
1670 PRINT "TE HAS CHOCADO CONTIGO MISMO"
1680 PRINT "QUE TE PASA. (ESTAS TONTO?"
1690 IF VV=1 THEN GOTO 2450
1700 LET VV=VV-1
1710 GOTO 550
1720 REM
1730 REM ************************
1740 REM * COLOCACION DE FRUTAS *
1750 REM ************************
1760 REM
1770 IF RND>.1 THEN RETURN
1780 LET V=INT(RND*4+1)
1790 LET X2=INT(RND*40+1)
1800 LET Y2=INT(RND*23+1)
1810 IF A(X2,Y2)<>0 THEN RETURN
1820 LET A(X2,Y2)=V+1
1830 LOCATE Y2,X2:PRINT F$(V)
1840 RETURN
1850 REM
1860 REM **************************
1870 REM * COMPROBACION DE CHOQUE *
1880 REM **************************
1890 REM
1900 IF A(X,Y)=0 THEN RETURN
1910 IF A(X,Y)=-1 THEN GOTO 1610
1920 IF A(X,Y)=1 THEN GOTO 1500
1930 LET SC=SC+15*A(X,Y):LO=A(X,Y)*3
1940 IF NC<>15 THEN LET NC=NC+1:RETURN
1950 CLS:LET NC=0
1960 PRINT "MUY BIEN. HAS TERMINADO CON LA FASE ";NV
1970 IF NV=4 THEN GOTO 2010
1980 PRINT "PREPARATE PARA LA SIGUIENTE"
1990 LET NV=NV+1
2000 GOTO 550
2010 PRINT "TAMBIEN HAS TERMINADO CON EL JUEGO"
2020 FOR I=1 TO 3000:NEXT I
2030 GOTO 2450
2040 REM
2050 REM *************************
2060 REM * MOVIMIENTO DEL GUSANO *
2070 REM *************************
2080 REM
2090 LET B$=INKEY$
2100 IF B$<>"P" AND B$<>"O" AND B$<>"Q" AND B$<>"A" THEN LET B$=C$
2110 LET C$=B$:LET X1=X:LET Y1=Y
2120 IF B$="P" THEN LET X=X+1
2130 IF B$="O" THEN LET X=X-1
2140 IF B$="Q" THEN LET Y=Y-1
```

```
2150 IF B$="A" THEN LET Y=Y+1
2160 GOSUB 1860
2170 LOCATE Y1,X1
2180 PRINT "o"
2190 LOCATE Y,X
2200 PRINT "O"
2210 LET A(X,Y)=-1
2220 IF LO=0 THEN GOTO 2250
2230 LET LO=LO-1
2240 RETURN
2250 LOCATE Y3,X3
2260 PRINT " "
2270 LET A(X3,Y3)=0
2280 IF A(X3+1,Y3)=-1 THEN LET X3=X3+1:RETURN
2290 IF A(X3-1,Y3)=-1 THEN LET X3=X3-1:RETURN
2300 IF A(X3,Y3-1)=-1 THEN LET Y3=Y3-1:RETURN
2310 IF A(X3,Y3+1)=-1 THEN LET Y3=Y3+1:RETURN
2320 RETURN
2330 REM
2340 REM *****************
2350 REM * (OTRA PARTIDA? *
2360 REM *****************
2370 REM
2380 PRINT:PRINT
2390 PRINT "(OTRA PARTIDA? (S/N)   "
2400 LET A$=INKEY$:IF A$="" THEN GOTO 2400
2410 IF A$="S" OR A$="s" THEN LET NV=1:LET VV=3:LET SC=0:GOTO 550
2420 IF A$="N" OR A$="n" THEN GOTO 2960
2430 GOTO 2400
2440 LET LO=LO-1
2450 REM
2460 REM *********************
2470 REM * TABLA DE GANADORES *
2480 REM *********************
2490 REM
2500 FOR I=1 TO 1000
2510 NEXT I
2520 CLS:PRINT "PUNTUACIONES DE HOY"
2530 PRINT:PRINT "----------------------------------------"
2540 FOR I=4 TO 17
2550    LOCATE I,1
2560    PRINT "!";TAB(40);"!"
2570 NEXT I
2580 PRINT "----------------------------------------"
2590 IF SC>=S(10) THEN LET S(10)=SC:LET A$(10)=CHR$(254)
2600 FOR I=1 TO 10
2610    FOR J=1 TO I
2620       IF S(J)<S(I) THEN LET A=S(J):LET S(J)=S(I):LET S(I)=A:LET B$=A$(J):LE
T A$(J)=A$(I):LET A$(I)=B$
2630    NEXT J
2640 NEXT I
2650 FOR I=1 TO 10
2660    IF A$(I)=CHR$(254) THEN LET PP=I
2670 NEXT I
2680 IF PP=0 THEN GOTO 2700
2690 A$(PP)="..............."
2700 FOR I=1 TO 10
2710    LOCATE 5+I,3
2720    PRINT A$(I);TAB(30);S(I)
2730 NEXT I
2740 LOCATE 5+PP,3,1
2750 LET A$(PP)=""
2760 IF PP=0 THEN GOTO 2850
2770 FOR I=1 TO 15
2780    LET A$=INKEY$:IF A$="" THEN GOTO 2780
2790 IF A$=CHR$(8) AND I>1 THEN PRINT CHR$(29);".";CHR$(29);:LET I=I-1:LET A$(PP
)=MID$(A$(PP),1,I)
2800 IF A$=" " THEN GOTO 2830
2810 IF A$=CHR$(13) THEN LET I=15:GOTO 2840
```

```
2820 IF A$<"0" OR A$>"z" THEN GOTO 2780
2830 PRINT A$;:LET A$(PP)=A$(PP)+A$
2840 NEXT I
2850 LOCATE 1,1,0
2860 LOCATE 20,1
2870 PRINT "<<< PULSA UNA TECLA >>>"
2880 LET A$=INKEY$:IF A$="" THEN GOTO 2880
2890 LOCATE 20,1
2900 GOTO 2330
2910 REM
2920 REM *****************
2930 REM * FIN DEL JUEGO *
2940 REM *****************
2950 REM
2960 CLS
2970 PRINT "HASTA PRONTO"
2980 PRINT "============"
2990 FOR I=1 TO 10
3000    PRINT
3010 NEXT I
3020 END
```

El programa es válido para todos los ordenadores, excepto el SPECTRUM. Las modificaciones que hay que hacer para que funcione en ordenadores distintos del IBM son:

COMMODORE:

```
180 PRINT CHR$(147)
370 PRINT CHR$(147)
610 PRINT CHR$(147)
1020 POKE 214, I-1:POKE 211, 7
1050 POKE 214, I-1:PODE 211, 32
1160 POKE 214, 4:POKE 211, I-1
1190 POKE 214, 18:POKE 211, I-1
1240 POKE 214, I-1:POKE 211, 7
1300 POKE 214, I-1:POKE 211, 7
1430 POKE 214, I-1:POKE 211, 18
1550 PRINT CHR$(147)
1660 PRINT CHR$(147)
1770 IF RND(1)>.1 THEN RETURN
1780 LET V = INT (RND (1)*4 + 1)
1790 LET X2 = INT (RND (1)*40 + 1)
1800 LET Y2 = INT (RND (1)*23 + 1)
1830 POKE 214, YS—1: POKE 211, X2—1: PRINT F$(V)
1950 PRINT CHR$(147): LET NC = 0
2090 GET B$
2170 POKE 214, Y1—1: POKE 211, X1—1
2190 POKE 214, Y—1: POKE 211, X—1
2250 POKE 214, Y3—1: POKE 211, X3—1
2400 GET A$: IF A$ = "" THEN GOTO 2400
2520 PRINT CHR$(147); "PUNTUACIONES DE HOY"
2550 POKE 214, I-1: POKE 211, 0
2560 PRINT "!"; TAB(39); "!"
2710 POKE 214, 4 + I: POKE 211, 2
2740 POKE 214, 4 + PP: POKE 211, 2
2780 GET A$: IF A$ = "" THEN GOTO 2780
2850 REM
2860 POKE 214, 19: POKE 211, 0
2880 GET A$: IF A$ = "" THEN GOTO 2880
2960 PRINT CHR$(147)
```

AMSTRAD:

```
1020 LOCATE 8,I
1050 LOCATE 33,I
1160 LOCATE I,5
1190 LOCATE I,19
1240 LOCATE 8,I
1300 LOCATE 8,I
1430 LOCATE 19,I
1830 LOCATE X2,Y2
2170 LOCATE X1,Y1
2190 LOCATE X,Y
2250 LOCATE X3,Y3
2550 LOCATE 1,I
2710 LOCATE 3,5 + I
2740 LOCATE 3,5 + PP,1
```

```
2860  LOCATE 1,20
2890  LOCATE 1,20
```

MSX:

```
1020  LOCATE 8,I
1050  LOCATE 33,I
1160  LOCATE I,5
1190  LOCATE I,19
1240  LOCATE 8,I
1300  LOCATE 8,I
1430  LOCATE 19.I
1770 IF RND (1)>.1 THEN RETURN
1780 LET V = INT (RND (1)*4 + 1)
1790 LET X2 = INT (RND (1)*40 + 1)
1800 LET Y2 = INT (RND (1)*23 + 1)
1830  LOCATE X2,Y2
2170  LOCATE X1,Y1
2190  LOCATE X,Y
2250  LOCATE X3,Y3
2550  LOCATE 1,I
2710  LOCATE 3,5 + I
2740  LOCATE 3,5 + PP,1
2860  LOCATE 1,20
2890  LOCATE 1,20
```

TECNICAS DE ANALISIS

DISEÑO DE SISTEMAS (III)



Especificación de los datos de entrada

LOS datos de entrada a un proceso pueden estar en varios «soportes» diferentes (elementos físicos sobre los que están contenidos los datos): primariamente, los datos estarán reflejados sobre un documento (papel) que haya sido relleno por alguna persona o impreso con alguna máquina: posteriormente estos datos se pasarán a un soprte magnético (cinta, diskette, disco...); en ocasiones, los datos de entrada a un proceso serán los de salida de otro anterior y podrán obtenerse sobre soporte magnético (y ya depurados, claro está).

En este último caso, en el diseño del proceso de que se trata poco habrá que especificar (excepto el formato de los datos) y, normalmente, la única tarea a realizar es reproducir los diseños preparados para el proceso anterior del que se obtienen los datos.

En cuanto a la preparación de los datos de entrada en su soporte primario (sobre papel), lo que nos interesa aquí es solamente subrayar que se debe prever espacio suficiente para la cómoda transcripción de los datos y que el diseño del impreso correspondiente debe tener en cuenta el orden y distribución que los datos tendrán en el fichero de entrada que se creará. Más adelante, abordaremos la problemática general de los documentos de soporte de datos (en cuanto a su racionalidad, optimización de espacio, orden para evitar errores, etcétera), por lo que aquí no profundizamos en este tema.

La situación más usual se produce cuando los datos contenidos en un documento cualquiera (no mecanizado) han de ser introducidos (por teclado) para la creación de un fichero de entrada. Es importante que, en el diseño del sistema, se analicen con cuidado y se describan estos ficheros, los «campos» que los componen, sus relaciones, etc., para evitar, en el momento de la programación, errores, dudas y pérdidas de tiempo.

El formato y características de los ficheros de entrada a un proceso se suelen reflejar en un documento como el que se presenta a continuación, a modo de ejemplo.

En él se suelen incluir los siguientes apartados:

N.° de fichero. Es una identificación codificada que complementa y concreta el nombre de fichero que se da en el encabezamiento del documento.

Utilidad. Breve descripción de la utilidad y contenido (desde el punto de vista funcional) del fichero de que se trata.

Soporte físico. Se indica el medio en que se presenta el fichero: cinta magnética, diskette, etc.

DESCRIPCION DEL FICHERO .... Pág. .../...

Fichero n.º	Soporte físico	Tamaño del fichero:	Nombre de registro ....
Utilidad	«Drive» a utilizar	N.º de regist. inicial	Frecuencia ....
....	N.º de volúmenes	N.º de regist. previsto	Volumen (promedio/máx.) ....
....	Secuencia de registros	% cambio	Tamaño reg. ....
....			Código ....

Ciclo ....

Disposición ....

Proc. seguridad ....

Programas que utilizan este fichero ....

Preparado por ....

Supervisado por ....

Aprobador por ....

Fecha ....

DESCRIPCION DE CAMPOS												
Nombre	Registro	Formato	Tamaño	Posición	Nivel	N.º repetición	V	Ceros	Signo	Verificación	Descripción

«Drive» a utilizar. Cuando el medio en que va grabado el fichero debe «montarse» en una unidad concreta.

N.º de volúmenes. Es importante el n.º de volúmenes que contiene (o puede llegar a contener) un fichero determinado, si es multivolumen, para su identificación y control y por razones de seguridad.

Secuencia de registros. Cuando en un fichero hay registros de varios tipos, conviene indicar la secuencia, si la hay, en que dichos registros se van a presentar. Además, los registros de un tipo concreto es usual que aparezcan en un orden predeterminado (quizá, incluso, clasificados) y puede ser útil conocer cuál es este orden.

Tamaño del fichero. Se suele indicar el n.º de registros previsto para el fichero, de tal modo que con el tamaño del registro en bytes —que se da a continuación— se puede establecer el tamaño total del fichero. A veces, en sucesivas versiones del fichero, o con su evolución en los posteriores procesos, el fichero crece: es útil, en este caso, dar una estimación del volumen máximo previsto. En otro apartado («% cambio» en el diseño que se da) se indica cómo va a evolucionar el tamaño del fichero.

Por otro lado, conviene describir cada tipo de registro de los que se pueden componer el fichero (en nuestro impreso se han previsto tres). Para cada registro se indican los siguientes datos:

Nombre de registro. Para su fácil identificación.

Frecuencia. Que, dependiendo de los casos, puede indicar la proporción de registros de este tipo que aparecen en el fichero (si éste tiene una composición bastante permanente en el tiempo) o la frecuencia con que estos registros aparecen en este fichero: puede haber datos diarios o datos que aparecen una vez a la semana en un fichero diario, etc.

Volumen. Volumen que representa este registro dentro del conjunto del fichero (porcentualmente o en valores absolutos).

Tamaño del registro. Se suele dar en «bytes».

Código. Clave o código con el que se puede distinguir a este registro dentro del fichero.

Además, conviene indicar (para el fichero en su conjunto) el ciclo de vida previsto y las condiciones bajo las que se puede disponer del fichero. Suele incluirse también una indicación de los procedimientos de seguridad y controles establecidos para este archivo. Se anota en el apartado correspondiente, el(los) progama(s) que utiliza(n) estos datos en su proceso.

El apartado más interesante del documento de definición de un fichero de entrada, es el que se destina a la descripción de cada uno de los campos que incluyen los registros del archivo.

Para cada campo se suelen especificar los siguientes apartados:

Nombre del campo correspondiente, para su identificación.

Registro al que pertenece, si hay varios en el fichero.

Formato; es usual utilizar un código (A: alfanumérico; B: binario; P: decimal empaquetado; U: decimal desempaquetado; E: en coma flotante; D: ídem en doble precisión, etc.) o bien directamente dar el formato con el que deberá ser representado en el programa correspondiente, según el lenguaje de programación utilizado.

Tamaño del campo (normalmente en bytes).

Posición dentro del registro (se da la posición del primer carácter del campo contando desde el comienzo del registro).

Nivel; n.° de nivel, si existe una estructura en niveles de los campos (un campo se subdivide en varios, cada uno de los cuales se puede subdividir, etc.).

N.° de repetición, si el campo se repite varias veces a su nivel, como sucede en ocasiones.

V; se suele marcar aquí si el n.° de repeticiones es variable.

Ceros; si el campo contiene ceros (en vez de blancos) cuando no tiene contenido.

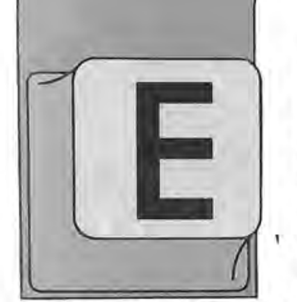
Signo; en los campos numéricos hay que indicar si el campo tiene signo y de qué tipo (en la 1.ª posición del campo por la izquierda; en la posición más a la derecha del campo; en la primera posición libre por la izquierda —signo flotante—, etc.).

Verificación. Conviene indicar si hay que hacer algún tipo de control o verificación sobre ese campo. En el documento correspondiente se reseña en detalle el tipo de verificación a establecer.

Descripción. Se reserva espacio para dar una descripción adicional del campo, o para incluir comentarios sobre su contenido, formato, etc.

TECNICAS DE PROGRAMACION

ASIGNACION DE VALORES A VARIABLES



EN los capítulos anteriores hemos descrito las estructuras de datos más frecuentes que aparecen en los programas de ordenador, así como los tipos de datos que se suelen utilizar. Ahora vamos a comenzar a describir qué es lo que se puede hacer con esos datos, es decir, las instrucciones ejecutables. Comenzaremos por la asignación de valores a variables.

Como se sabe, una variable no es otra cosa que un nombre al que se puede asignar uno o varios valores, dispuestos en una estructura determinada. Normalmente, en todos los lenguajes de programación, los nombres de las variables no pueden elegirse sin restricción alguna, sino que es preciso someterse a ciertas reglas, que en general son lo bastante amplias como para que el número de nombres diferentes que se pueda construir sea prácticamente ilimitado.

En BASIC, por ejemplo, el nombre de una variable debe comenzar por una letra, que puede o no ir seguida por cierto número de letras, cifras o el punto decimal, pudiendo terminar en uno de los cuatro caracteres especiales de definición de tipo: $,%,!,#. Las variables que terminan en $ tienen automáticamente el tipo «string» y contienen datos literales. Las que terminan en % sólo pueden adoptar valores enteros. Las que terminan en ! pueden tomar valores con decimales en precisión normal. Las que terminan en # tomarán valores con decimales en precisión doble. Finalmente, las que no terminan en ninguno de estos caracteres suelen ser, por omisión, variables reales que adoptan valores con decimales de precisión normal, a menos que se haya utilizado alguna de las instrucciones de definición de tipo (DEFINT, DEFDBL o DEFSTR) mencionadas en los capítulos anteriores.

El número de caracteres que puede tener el nombre de una variable BASIC es variable, siempre que se mantenga dentro de ciertos límites, que dependen del intérprete o compilador utilizado. En el intérprete de BASIC del IBM PC, por ejemplo, el nombre de las variables puede tener hasta 40 caracteres. Esto significa que el número de nombres distintos que se puede definir es prácticamente infinito y rebasa todas las posibilidades prácticas de un programa. Además, las letras minúsculas se consideran equivalentes a las mayúsculas y se traducen a ellas automáticamente.

En otros intérpretes, el límite es más pequeño. En el SPECTRUM, por ejemplo, el número máximo de caracteres en el nombre de una variable es igual a 2. En el caso de las variables literales, como el segundo carácter del nombre tiene que ser un dólar, sólo podremos elegir 26 nombres diferentes en un programa determinado. Con las variables de valor numérico, en cambio, dispondremos de 26 × 26 posibilidades para los nombres de dos letras, pues los dos caracteres del nombre pueden ser letras. Además, podremos definir también 26 nombres de una sola letra, lo que hace un total de

702 nombres diferentes. Hay otra restricción adicional: si la variable numérica es un vector o una matriz (una serie o una tabla), su nombre sólo podrá tener una letra, por lo que sólo podrá haber 26 nombres diferentes para estas estructuras. Como se ve, estos números son lo bastante pequeños como para que sea posible que al construir un programa para el SPECTRUM nos quedemos sin nombres de variables de algún tipo o estructura.

Veamos algunos ejemplos de nombres válidos de variables BASIC, junto con el tipo al que pertenecen:

Variable	Tipo
A	real normal
AB!	real normal
ABC#	real doble
A12.B	real normal
C$	literal
C1.2$	literal
I1%	entero

De todos los nombres anteriores, sólo el primero y el quinto serán válidos en el SPECTRUM. Todos ellos serán correctos, sin embargo, en la mayor parte de los intérpretes y compiladores del lenguaje BASIC. En cambio, los siguientes nombres son incorrectos en cualquier traductor de dicho lenguaje, pues no cumplen las condiciones generales indicadas más arriba:

1A $C ABC/ ¡HOLA!

En PASCAL, los nombres de variables deben empezar por una letra, que puede o no ir seguida por un número variable de letras o cifras y a veces por el carácter de subrayado. La longitud máxima permitida varía según el compilador, aunque suele reducirse a 8 caracteres. Los nombres más largos no son inválidos. Simplemente se ignoran los caracteres sobrantes. Existe, sin embargo, la restricción de que ciertas «palabras reservadas» no pueden utilizarse como nombres de variables. Mencionaremos las siguientes, entre otras muchas:

array and not or to goto if begin end

Son, por tanto, nombres válidos en PASCAL, los siguientes:

Juan caso1 caso_uno

Como se ve en el primer ejemplo, muchos compiladores admiten que se mezclen letras mayúsculas y minúsculas en el mismo nombre. En cambio, son nombres incorrectos los siguientes:

1A $C ABC/ ¡HOLA!

En APL, los nombres de variables pueden comenzar con una letra y continuar o no con letras o cifras. Muchas veces se admiten también los caracteres de subrayado y super-rayado, aunque no en primer lugar. Las letras minúsculas son distintas de las mayúsculas, por lo que los nombres «a» y «A» corresponderán a variables diferentes. También serán distintas las cuatro variables siguientes:

AB Ab aB ab

El número máximo de caracteres en un nombre varía según el intérprete. En el caso del intérprete APL/PC de IBM, este número debe ser menor o igual que 12. Los nombres más largos son admitidos, pero los caracteres sobrantes se ignoran.

Además de nombre, las variables tienen valor. Sin embargo, así como el nombre es intrínseco a la variable, el valor puede cambiar de un momento a otro. (Es por esta razón, precisamente, por lo que se las llama «variables»). Por tanto, debe haber alguna manera de cambiar dicho valor o dárselo por primera vez. Las instrucciones que realizan esta operación se denominan «instrucciones de asignación», que existen en todos los lenguajes de programación de alto nivel y tienen una forma muy semejante en todos ellos.

Una instrucción de asignación tiene siempre la siguiente estructura:

variable símbolo valor

donde «variable» es el nombre de la variable; «símbolo» es un carácter o grupo de caracteres que indica que queremos realizar una asignación de valor; finalmente, a la derecha de dicho símbolo se coloca el valor o el conjunto de valores que deseamos asignar a la variable.

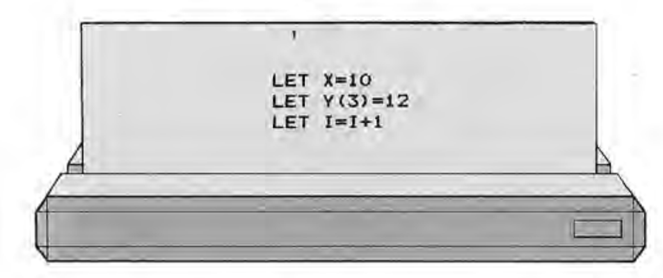
En BASIC, el símbolo de las instrucciones de asignación es el signo igual (=). Además, se añade a la izquierda de la instrucción de asignación la palabra reservada LET. Finalmente, el valor que se le asigna a la variable debe ser un escalar o un «string» (es decir, sólo puede asignarse un valor único o una cadena de caracteres en cada instrucción de asignación). Resumiendo: la instrucción de asignación BASIC tiene la forma general siguiente:

LET variable = valor

La palabra LET es exigida por algunos intérpretes (como el del SPECTRUM), pero puede ser omitida en otros (la mayoría) por lo que la instrucción puede reducirse, casi siempre, a:

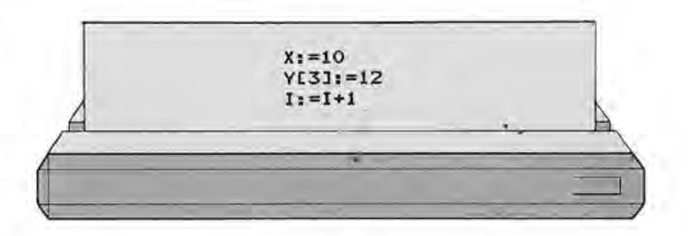
variable = valor

Veamos algunos ejemplos:

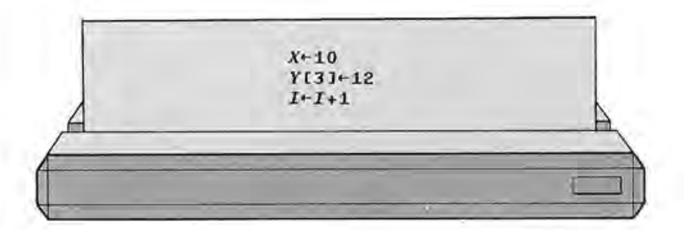


Obsérvese que, si estamos utilizando uno de los intérpretes de BASIC que permiten eliminar la palabra LET, la tercera instrucción de nuestro ejemplo toma una forma muy rara: «I=I+1». Cualquiera que tenga unas nociones elementales de álgebra comprende que, si esta expresión fuera una expresión algebraica, no tendría solución, pues no existe ningún número que sea igual a la suma de sí mismo más uno. En realidad, lo que ocurre es que el signo igual en BASIC no tiene la misma significación que en el álgebra. La instrucción anterior no quiere decir que I sea igual a I+1, sino que tomamos el valor que tenía I (cualquiera que sea), le sumamos uno, y el resultado se lo asignamos a I como nuevo valor, que sustituye al anterior. No es difícil comprender esto, pero puede costarle cierto trabajo al principiante, y hace necesaria esta explicación. Por esta razón, otros lenguajes, como PASCAL y APL, han procurado eliminar el problema eligiendo otro símbolo, distinto del signo igual, para representar la asignación de valor a una variable.

En PASCAL, por ejemplo, se utiliza como símbolo de asignación la combinación de los dos puntos con el signo igual. Las mismas asignaciones que vimos más arriba para el caso del BASIC se expresarían en PASCAL así:



En APL el símbolo de asignación es una flecha dirigida hacia la izquierda. Veamos cómo se expresarían en este lenguaje las asignaciones anteriores:



Los ejemplos anteriores nos permiten ver cómo se realiza la asignación de valores a las variables cuando se trata de un valor único. Incluso vemos cómo se puede realizar la asignación a uno solo de los elementos de una serie o vector. Pero ¿qué ocurre si queremos asignarle valor a todas y cada una de las componentes de una serie o de una tabla de datos?

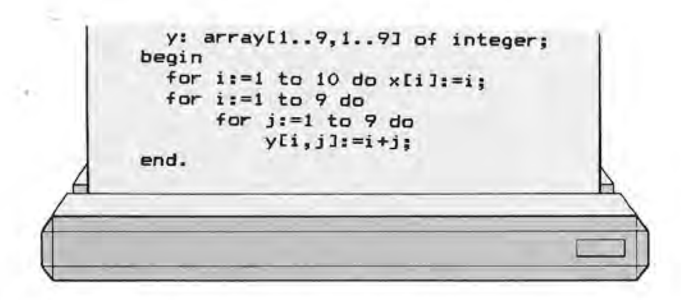
En BASIC, por ejemplo, tenemos que escribir uno o varios bucles, con objeto de que la asignación se realice siempre elemento a elemento. Supongamos que queremos definir una serie de diez datos cuyo nombre será X y cuyos valores han de ser los números del 1 al 10. Supongamos que también queremos construir una tabla llamada Y, de nueve filas y nueve columnas, con los valores de la tabla de sumar, es decir, tal que el valor del elemento (i; j) sea igual a (i+j). El siguiente programa lo conseguiría:

```
10 DIM X(10), Y(9,9)
20 FOR I=1 TO 10
30 LET X(I)=I: NEXT I
40 FOR I=1 TO 9
50 FOR J=1 TO 9
60 LET Y(I,J)=I+J
70 NEXT J: NEXT I
```

Obsérvese el programa anterior: la instrucción 10 define las dimensiones de la serie X y de la tabla Y. Las instrucciones 20 y 30 forman un bucle que utiliza como índice la variable escalar I para asignar el valor deseado a cada uno de los elementos de X. Finalmente, las instrucciones 40, 50, 60 y 70 forman un doble bucle que asigna a cada elemento de la tabla Y el valor correspondiente a la tabla de sumar.

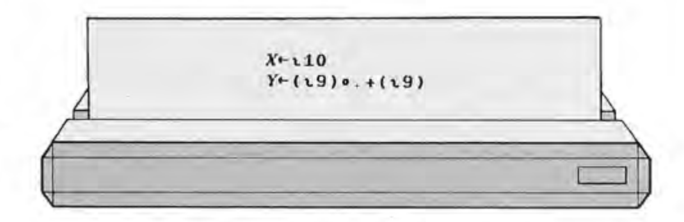
En PASCAL, un programa equivalente al anterior sería muy semejante. Veámoslo:

```
program TABLAS;
var
  i: integer;
  x: array[1..10] of integer;
```



```
  y: array[1..9,1..9] of integer;
begin
  for i:=1 to 10 do x[i]:=i;
  for i:=1 to 9 do
      for j:=1 to 9 do
          y[i,j]:=i+j;
end.
```

En APL, sin embargo, es posible asignar a una serie o tabla todos los datos que debe contener con una sola instrucción. Veamos cómo se realizarían las dos asignaciones anteriores:



```
X←⍳10
Y←(⍳9)∘.+(⍳9)
```

Como se recordará por capítulos anteriores, la primera instrucción asigna a X la serie de los números de 1 a 10. La segunda asigna a Y la tabla de sumar de nueve filas y nueve columnas.

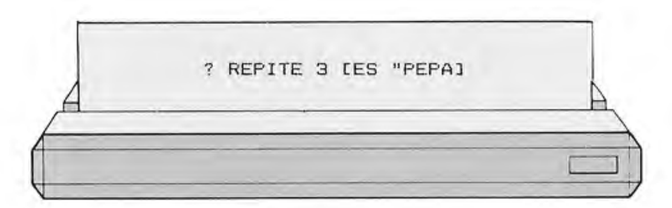
LOGO



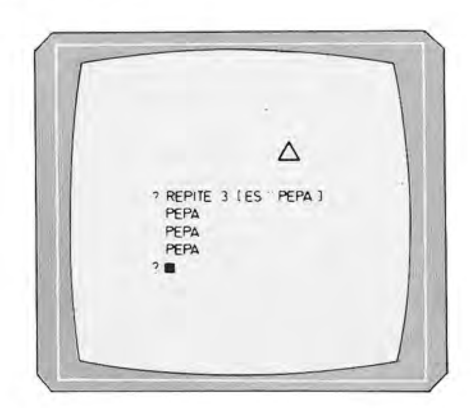
Jugando con el comando de repetición

HASTA este momento, el conjunto de comandos que hemos puesto dentro de la lista del comando REPITE han sido avanzar y girar. Hay que tener en cuenta que en esta lista podemos poner cualquier palabra (comando) que la tortuga conozca, incluso otro REPITE. Vamos a ver varios ejemplos.

Supongamos que queremos escribir nuestro nombre varias veces en la pantalla. La orden sería:



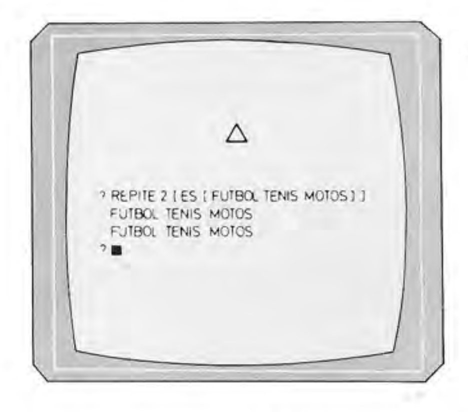
y nos quedaría:



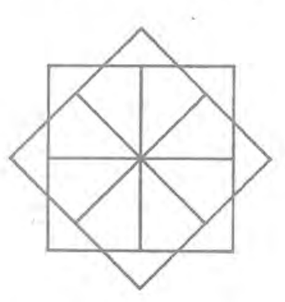
También podemos escribir varias veces los nombres de nuestros deportes favoritos. El comando sería:



y obtendríamos como resultado:

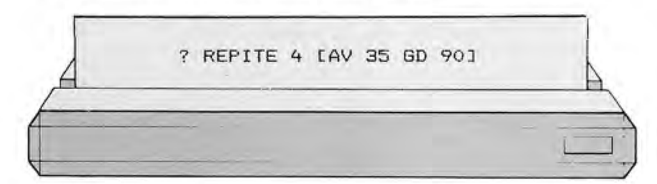


Ahora vamos a dibujar la siguiente figura:

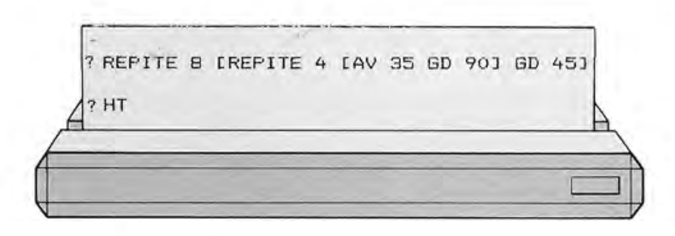


Como vemos, está formada por 8 cuadrados que van estando girados cada uno respecto al anterior. Como sabemos,

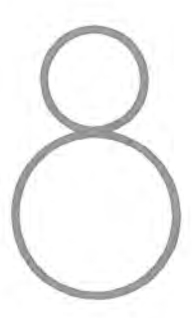
la orden para dibujar un cuadrado es:



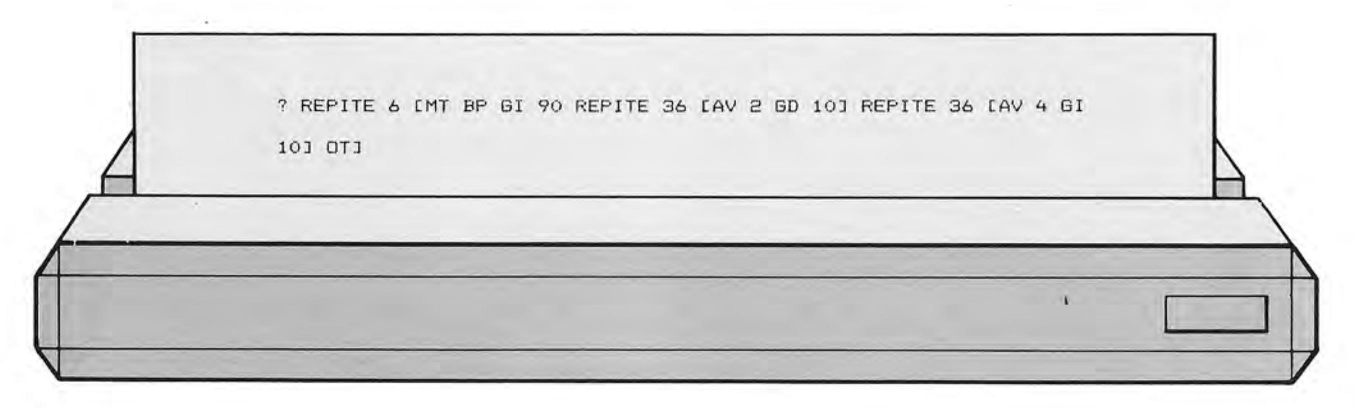
Luego para hacer la figura las órdenes serían:



Si ahora deseamos pintar 2 circunferencias de esta forma:



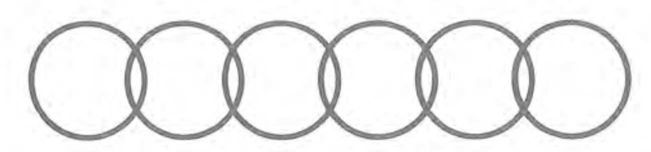
de manera que aparezcan y desaparezcan varias veces, escribiríamos lo siguiente:





Os proponemos

1. Haz el siguiente dibujo, teniendo en cuenta que cada circunferencia está desplazada con respecto a la anterior.



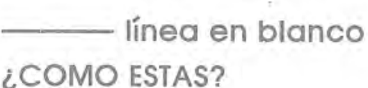
2. Pinta una figura parecida a la de los 8 cuadrados girados, pero que tenga en su lugar 12 pentágonos.

3. Sabiendo que para dejar una línea en blanco (sin que la tortuga escriba nada) hay que poner:

ES [ ]

dale las órdenes necesarias para que escriba lo siguiente:

¿COMO ESTAS?
BIEN, GRACIAS

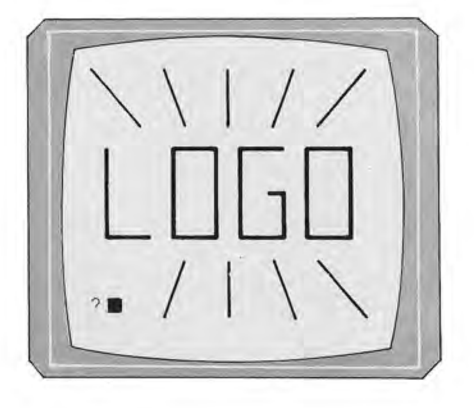


——— línea en blanco

¿COMO ESTAS?
BIEN, GRACIAS

——— línea en blanco

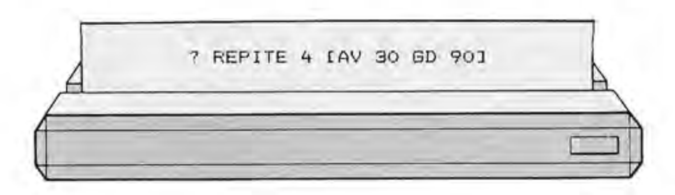
4. Haz que la tortuga pinte la palabra LOGO en la pantalla, y que aparezca y desaparezca varias veces.



Cómo enseñar cosas nuevas a la tortuga

Hasta ahora la tortuga sólo entiende y ejecuta aquellas órdenes que le escribimos usando los comandos. Pero nuestra tortuga también es capaz de aprender palabras nuevas y saber lo que significan para después ejecutarlas.

Así, por ejemplo, cada vez que queremos que pinte un cuadrado hemos de decirle:



? REPITE 6 (AV 30 GD 90)

pero tenemos la posibilidad de enseñarle a la tortuga cómo se hace un cuadrado.

Para enseñarle a la tortuga a hacer cosas nuevas se utiliza el comando:

PARA "nombre

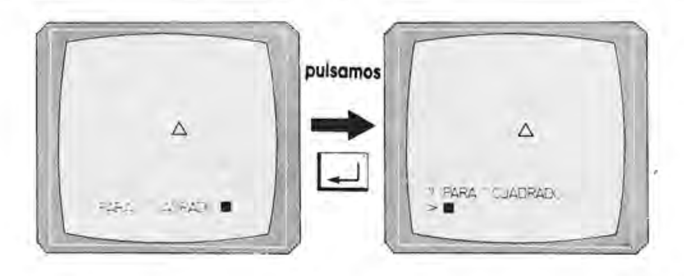
donde **nombre** es una palabra nueva para la tortuga, pero que a partir de ahora va a entender.

A continuación hemos de poner un conjunto de comandos que la tortuga ya conozca y que le diga cómo se hace esa cosa nueva.

Por último, para decirle que ya hemos terminado de enseñárselo hemos de poner:

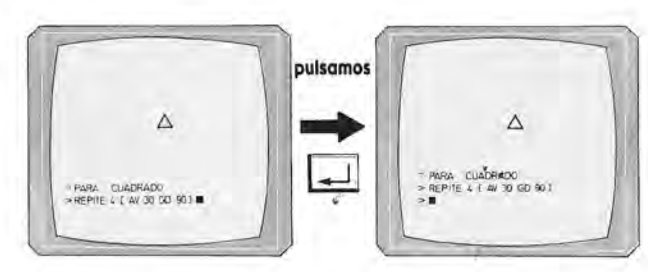
FIN

Ahora vamos a enseñar a la tortuga a hacer un cuadrado:



Como vemos, después de pulsar la tecla RETURN, en lugar de salir el signo de interrogación (?) que indica que la tortuga está preparada para recibir órdenes, nos sale un signo de mayor (>). Este signo sirve para decirnos que la tortuga está esperando a que la enseñemos a hacer algo, en nuestro caso un cuadrado.

Por tanto, ponemos:



y como ya no se necesitan más comandos para hacer un cuadrado, escribimos FIN para decirle que ya hemos acabado:



En este momento, tras pulsar RETURN, la tortuga nos indica que ya ha aprendido a hacer un cuadrado escribiéndonos un mensaje en la pantalla:

CUADRADO DEFINIDO

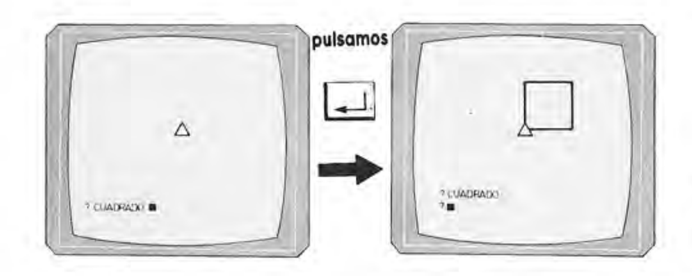
o,

ACABAS DE DEFINIR CUADRADO

y diciéndonos que espera que le demos una orden (aparece la interrogación).

Al conjunto de comandos o palabras que le enseñan a la tortuga cómo hacer algo se le llama PROCEDIMIENTO, ya que un procedimiento es un método o serie de reglas para realizar alguna cosa.

Pues bien, una vez definido nuestro procedimiento CUADRADO, vamos a ver si, efectivamente, la tortuga sabe hacerlo o no. Para ello, escribimos:



Ahora tú puedes enseñar a la tortuga muchas de las figuras que has dibujado antes. La tortuga las aprenderá y no las olvidará hasta que apagues el ordenador.

Por ejemplo, vamos a hacer que la tortuga aprenda a dibujar estrellas. Si escribimos:



nos quedará:



La tortuga ha desaparecido de la pantalla porque al enseñarle a dibujar la estrella le hemos dicho que se oculte (comando OT).



Cómo hacer que la tortuga olvide cosas

Si al definir un procedimiento, es decir, enseñar a la tortuga a hacer una cosa nueva para ella y mandarle que lo ejecute nos damos cuenta de que no dibuja lo que nosotros queremos, significa que nos hemos equivocado al escribir algo y tenemos que cambiarlo.

Por ahora, lo que vamos a hacer es decirle que se olvide de eso nuevo que le hemos enseñado y definir el procedimiento otra vez.

Para borrar un procedimiento, es decir, que la tortuga se olvide de cómo se hace, usaremos el comando:

BORRA "nombre

o en abreviatura

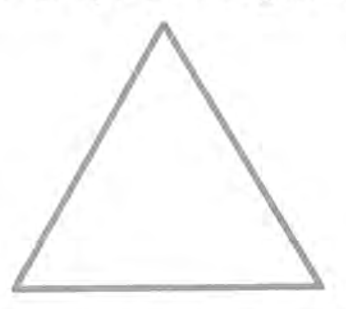
BO "nombre

siendo **nombre** el nombre del procedimiento que queremos borrar.

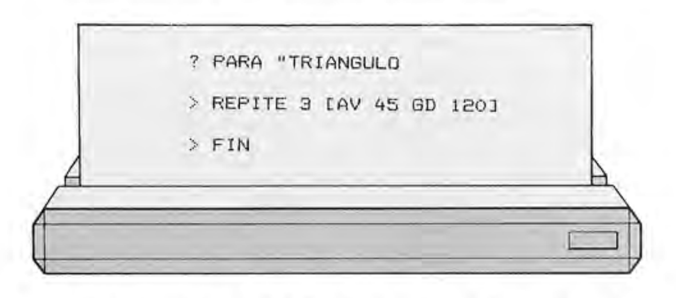
Si en lugar de querer borrar un solo procedimiento, deseamos borrar varios pondremos sus nombres en una lista:

BO (nombre1 nombre2 ... nombren)

Veamos un ejemplo. Supongamos que queremos dibujar un triángulo así:



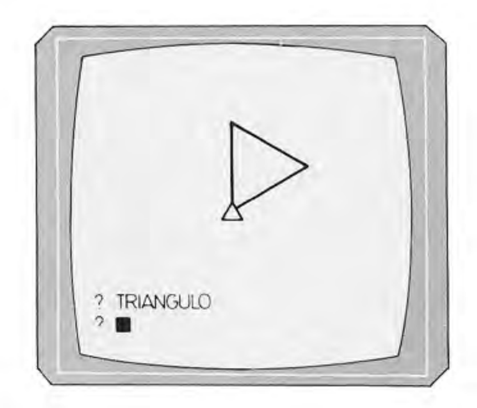
Si definimos un procedimiento:



y mandamos a la tortuga que lo ejecute:

? TRIANGULO

nos saldrá:



que no es lo que nosotros queremos.

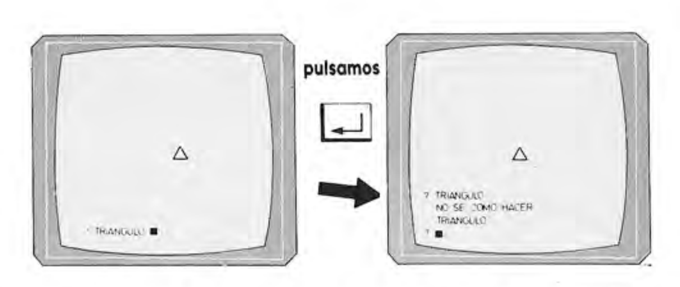
Por tanto, nuestro procedimiento no nos vale y tenemos que borrarlo de la memoria de la tortuga:



Para comprobar que, efectivamente, lo ha olvidado si le decimos ahora:

? TRIANGULO

nos responderá con un mensaje de error:



Entonces, en este momento podemos enseñarle de nuevo a hacer el triángulo, pero correctamente:

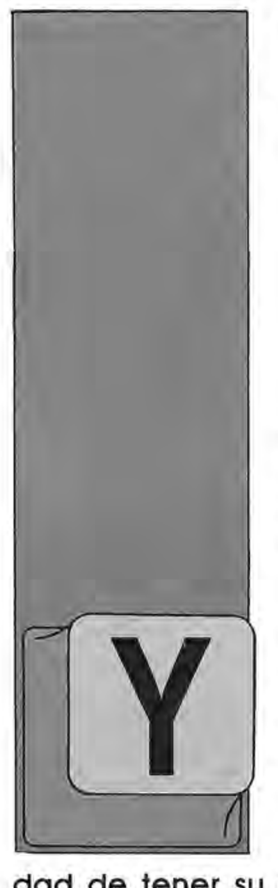


Por último, si en lugar de querer borrar uno o varios procedimientos, deseamos eliminar todos, no hace falta dar cada uno de sus nombres, sino que podemos utilizar el comando

BOTODO

que le indica a la tortuga que se olvide de todas las cosas nuevas que ha aprendido.

PASCAL



Variables locales de un procedimiento

Ya hemos mencionado que la estructura de los procedimientos y funciones es muy similar a la de los programas, y que entre esas similitudes se encuentra la posibilidad de tener su propia zona de definición de datos «locales».

Las variables locales de un procedimiento son iguales en casi todos los aspectos a las del programa principal, denominadas a su vez «globales» por poder utilizarse en cualquier zona del programa, sea la parte principal, un procedimiento o una función. Solamente hay dos diferencias:

1. Las variables locales de cada procedimiento sólo se pueden utilizar dentro de él (y de los procedimientos que éste pudiera a su vez tener dentro); para el programa principal y para los demás procedimientos es como si no existieran. Esto se cumple también con todas las demás cosas definidas localmente dentro del procedimiento.

2. Sólo toman la porción de memoria que necesitan en el momento en que se utiliza el procedimiento. Cuando se termina de ejecutar éste, el espacio de memoria ocupado por sus variables queda libre para cualquier otra cosas, como, por ejemplo, para las variables locales de otro procedimiento que se ejecute a continuación. Por ello, no es posible conservar datos en una variable local entre dos utilizaciones de un procedimiento. Una vez se sale de éste, se pierden todos sus datos locales. Si se necesitara guardar algún dato entre ejecuciones sucesivas habría que utilizar variables globales. Estas últimas se dice que son variables «estáticas», pues tienen su espacio de memoria asignado permanentemente; por el contrario, las variables locales se dice que son «dinámicas».

(*NOTA:* En otros lenguajes como el Modula-2 o el PL1, es posible tener variables locales estáticas, e incluso esta posibilidad existe en algunas versiones de PASCAL como, por ejemplo, con Turbo-Pascal por medio de las denominadas «typed constants».)

Si cuando se está traduciendo la zona correspondiente a un procedimiento, el compilador encuentra el nombre de una variable, busca primero a ver si la encuentra entre las definidas localmente; de encontrarla, es ésa la que utiliza. Sólo si no la encuentra pasa a buscar entre las variables globales del programa.

Por ello, si una variable local tiene el mismo nombre que una global, la que se utilizará siempre en el procedimiento será la variable local. Al traducir la zona correspondiente al programa principal, no hay duda posible, pues la local es como si no existiera.

(Cuando se trata de un procedimiento, dentro de un procedimiento, dentro de otro, etc., busca primero entre las locales propias; si no la encuentra, busca entre las del procedimiento de orden superior —que a su vez son globales para el anterior—, y así sucesivamente hasta acabar buscando entre las del programa principal.)

El uso de variables locales permite ahorrar memoria, pues las mismas zonas son utilizadas por variables de diferentes procedimientos, según el momento. Sin

embargo, las principales ventajas son otras:

— Por un lado, evitan que, si el procedimiento altera (erróneamente o no) el contenido de una variable, esto afecte al resto del programa, cosa que podría suceder si la variable fuera global.

— Por otro lado, permiten escribir procedimientos autosuficientes, es decir, que no necesitan que se les prepare en la zona de definición de datos del programa ninguna variable especial para ellos. Así, procedimientos escritos y probados en alguna ocasión para otros programas pueden ser utilizados sin más que copiar su descripción. De esta manera se ahorra una enorme cantidad de trabajo y se evitan posibles errores.

El programa que viene a continuación es similar al último que vimos, pero con un procedimiento para la fase de cuenta:

```
program Contar;

(*-----------------------------------*)
procedure Contar1a10;
  var N: integer; (* Esta variable es local *)
 begin
  for N:= 1 to 10 do writeln (N:3)
 end;

procedure Presentacion;
 begin
  clrscr;   (* o PAGE  *)
  writeln ('Esto cuenta de 1 a 10.');
  writeln ('Y si no se lo creen, miren:')
 end;

procedure Despedida;
 begin
  writeln ('Yo casi nunca miento. Adiós.')
 end;
(*-----------------------------------*)

begin  (* Aquí está el programa principal *)
 Presentacion;
 Contar1a10;
 Despedida
end.
```

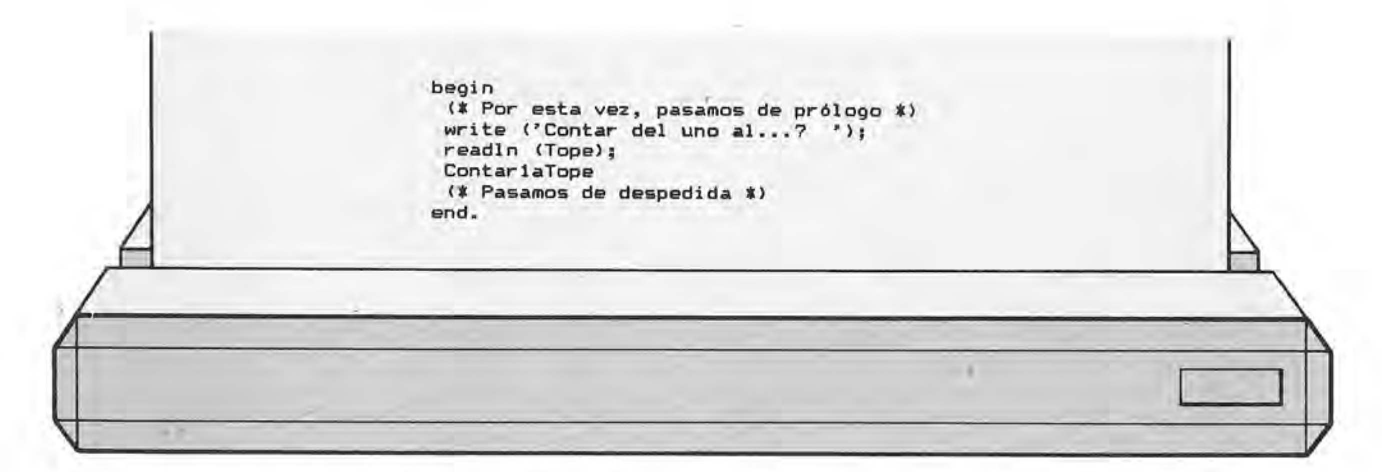
Paso de parámetros por valor

Si se necesitase que un procedimiento manejara datos pertenecientes al programa principal (o al procedimiento en que estuviera inserto, caso de que fuese de menor nivel), una posible forma sería utilizar la o las variables globales en que estuvieran guardados esos datos llamándolas por su nombre y cuidando, por supuesto, de que no hubiese variables locales con igual denominación.

Partiendo del programa **Contar,** vamos a hacer otro que realice una cuenta desde 1 hasta un valor introducido por teclado en la variable global TOPE.

```
program CuentaVariable;
 var Tope:  integer;

procedure Contar1aTope;
  var N:  integer; (* Esta variable es local *)
 begin
  for N:= 1 to Tope do writeln (N:3)
 end;
```



Si embargo, este método obliga a que el programa principal y el procedimiento se pongan de acuerdo en la variable a utilizar, como sucede con Tope, y ésta tendrá que llamarse igual dentro y fuera del procedimiento. Así, la «autosuficiencia», o facilidad de utilizar en un nuevo programa procedimientos que se hicieron en otro momento sin más que copiarlos, resulta bastante menor.

Además, no podríamos suministrar al procedimiento los resultados de expresiones ni los datos contenidos en otras variables diferentes a las convenidas sin guardarlos previamente en éstas cada vez que hubiera que ejecutarlo.

Por otra parte, tampoco se pueden utilizar variables locales normales para pasar los datos, pues ya se ha dicho que el programa principal no tiene acceso a ellas; al existir éstas en memoria sólo durante la ejecución del procedimiento, no sería posible guardar los datos en ellas justo antes de ejecutarlo.

Para solucionar esto, existe en PASCAL la posibilidad de transferir datos al tiempo que se llama al procedimiento por medio de ciertas variables locales especiales.

Para ello, en su cabecera y a continuación del nombre se deben definir las variables locales en las que se van a guardar los datos a transferir en el momento de la llamada, describiendo lo que se denomina «lista de parámetros».

Esto se hace de manera idéntica a como se definen las variables en la zona de descripción de datos, pero poniendo el conjunto de definiciones entre paréntesis. Cabeceras válidas serían por tanto:

```
procedure Koke  (Ene: integer);
procedure Bibi  (Max: integer; C: char);
procedure Gon   ( I,J,K: integer;
                  B: boolen );
```

Las variables descritas en la lista de parámetros son unas variables locales más, con la única particularidad de que se guardan datos en ellas justo en el momento de llamar al procedimiento.

Los datos a guardar se deben indicar entre paréntesis y separados entre sí por comas, a continuación del nombre del procedimiento, cada vez que se utilice éste. Deben ir exactamente en el mismo orden en que están definidas las variables en la cabecera. En la lista de parámetros a transferir pueden aparecer tanto constantes como variables o expresiones siempre que el tipo coincida con el indicado en la cabecera. Por ello:

```
Koke (1)
```

haría que Ene valiera 1 nada más comenzarse a ejecutar Koke; con

```
Bibi (5 + 4, succ('A'))
```

Max valdría 9 y C valdría 'B'. Por último,

```
Gon (Tope, Cuenta, 3, 2 = 3)
```

sería también correcto y con ello I y J tomarían los valores de Tope y Cuenta en el momento de la llamada, K valdría 3 y B sería FALSE.

Sin embargo,

```
Koke (true) y Bibi (13)
```

producirían errores en la compilación pues Bibi precisa dos parámetros y el tipo del parámetro de Koke es INTEGER.

Escribamos un programa que haga diferentes cuentas, cada vez más largas:

```
Program VariasCuentas;
 var Tope: integer;

procedure ContarHasta (Final: integer);
(* Cuenta desde 1 hasta el valor de Final *)
  var N: integer;
 begin
  for N:= 1 to Final do write (N:4)
 end;

begin
 for Tope:= 1 to 10 do
  (* Contar con limite creciente: *)
   begin
    ContarHasta (Tope);
    writeln
   end;

 writeln ('Y ahora la buena:');
 ContarHasta (1000)
end.
```

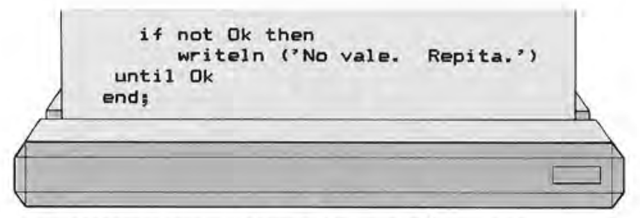
Aquí se observa una de las ventajas más importantes de los procedimientos:

Si el procedimiento ContarHasta hubiera sido escrito por otro programador, nosotros no necesitaríamos conocer absolutamente nada de sus interioridades para utilizarlo. Bastaría con copiarlo y saber, primero, que se llama ContarHasta y, segundo, que precisa de un único parámetro de tipo INTEGER que indica el final de la cuenta. Nada más. Por ello es una buena costumbre describir con un comentario para qué sirve un procedimiento al lado de su cabecera.

Paso de parámetros por nombre

Supongamos ahora que queremos escribir un procedimiento que lea de teclado la edad de una persona, comprobando que no es errónea, y la guarde en la variable que se desee. En principio podría ser algo como:

```
procedure LeeEdad (E: integer);
   var Ok:  boolean;
 begin
  write ('Edad:  ');
  repeat
    readln (E);
    Ok:= (0 <= E) and (E <= 120);
```



```
      if not Ok then
         writeln ('No vale.  Repita.')
    until Ok
   end;
```

Entonces para utilizarlo haríamos:

LeeEdad (EdadPadre);
LeeEdad (EdadHijo);

Sin embargo, debemos recordar que la variable E descrita en la cabecera es local y que, por tanto, por mucho que cambiemos su valor, EdadPadre y EdadHijo permanecerán inalteradas.

Podríamos escribir LeeEdad de manera que él mismo guardara el dato en la variable EdadPadre poniendo, por ejemplo, EdadPadre:=E como última instrucción, pero está claro que entonces sólo serviría para esa variable.

Para solucionar estas situaciones está lo que se denomina transferencia de parámetros por nombre.

Utilizando esta modalidad, las variables descritas en la cabecera no son locales del procedimiento y cada vez que se ejecuta éste, la porción de memoria que les corresponde es precisamente la de las variables que aparecían en la lista de parámetros en el momento de la llamada.

En otras palabras, la variable de la cabecera es en cada llamada exactamente la misma que figura en la lista de parámetros, de manera que si modificamos su valor modificamos también el de esta última.

Para indicar que un parámetro se transfiere por nombre, se debe poner delante de su nombre la palabra reservada VAR:

```
Procedure Aimu (      I:  integer;
                var J,K:  integer;
                      B:  boolean;
                  var C:  char       );
```

en este procedimiento, J,K y C se transfieren por nombre y los demás por el método habitual.

Evidentemente, cuando un dato se transfiere por nombre, sólo puede aparecer una variable en la lista de parámetros; no están permitidas ni constantes ni expresiones. Veamos un ejemplo:

```
program LLamadaPorNombre;
 var Numero: integer;

 procedure PoneraCero (var N: integer);
 (* Pone a cero la variable transferida *)
 begin
  writeln ('La variable transferida valía ',N);
  writeln ('Pero desde ahora valdrá 0');
  N:= 0
 end;

begin
 Numero:= 7; (* Numero ahora vale 7 *)
 writeln ('Número=',Numero);
 PoneraCero (Numero);  (* Pero ahora vale 0 *)
 writeln ('Número=',Numero)
end.
```

Si quitáramos ahora la palabra VAR de la cabecera de PoneraCero, comprobaríamos que Número no cambia de valor. Por otra parte, si en el programa pusiésemos la instrucción PoneraCero (2+2), se produciría un error al compilar, cosa que no sucedería si no estuviera la palabra VAR.

Por tanto, para que el procedimiento LeeEdad funcionara correctamente bastaría con que su cabecera fuese la siguiente:

```
procedure LeeEdad (var E: integer);
```

Hay una ventaja adicional en el uso del paso de parámetros por nombre que debe utilizarse con sumo cuidado.

Cuando el dato a transferir ocupa mucha memoria (dentro de poco estudiaremos tipos de datos que pueden hacerlo), al utilizar el procedimiento por el método habitual se crea una variable local de las mismas dimensiones, con lo que las necesidades de memoria son muy grandes. Además, cada vez que se utiliza se produce una transferencia automática de los datos a la variable local y como son muchos el programa puede llegar a hacerse más lento.

Pasando los datos por nombre, la porción de memoria que se utiliza es la misma del original y, por tanto, no se necesita tanta memoria ni emplear tanto tiempo en copiar los datos. Eso sí, existe el riesgo de que por una mala programación alguno de los datos originales resulte alterado de manera involuntaria al ejecutarse el procedimiento.



OTROS LENGUAJES

SISTEMAS OPERATIVOS: PICK

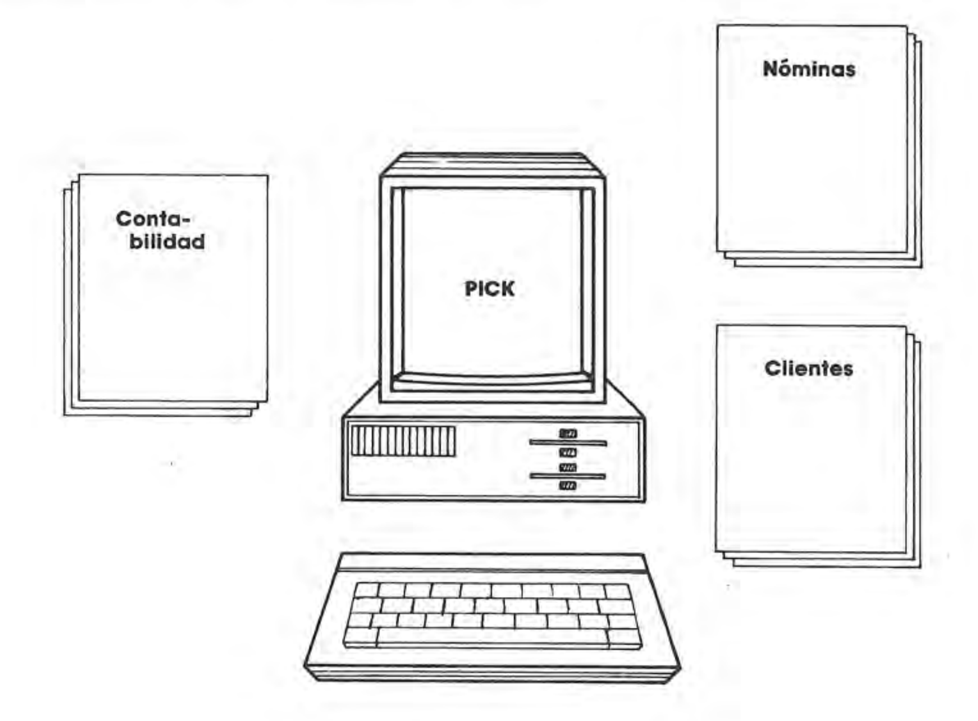


Introducción

EL sistema operativo PICK es un sistema operativo multiusuario que está orientado a la gestión de información que funciona en tiempo real. Entre sus características más sobresalientes está la posibilidad de multiprogramación (ejecución de varios programas al mismo tiempo) y la gestión de memoria virtual.

En varios aspectos se parece a UNIX, ya que ambos fueron desarrollados originalmente para miniordenadores, y están emigrando al entorno de los microordenadores. La primera versión comercial de PICK para miniordenadores apareció en 1973, y en 1979 se dispuso de la primera versión para un microordenador.

Tanto UNIX como PICK son muy potentes, a la vez que complicados, ya que proporcionan un gran número de comandos y utilidades. Sin embargo, PICK difiere de UNIX en su entorno de trabajo. Mientras que el primero está mejor adaptado para las aplicaciones de gestión, el segundo se concibió como una herramienta para el desarrollo de software y, por tanto, es más adecuado para aplicaciones de tipo científico, así como de ingeniería.





Las tareas de gestión propias de una empresa se pueden mecanizar gracias al sistema operativo PICK.



Estructura general del sistema

El «núcleo» de PICK es su base de datos relacional, en torno a la cual funciona todo el sistema. Además, dispone de los siguientes elementos:

1. Un lenguaje de consulta de acceso a la base de datos, llamados ACCES. En este lenguaje las órdenes reciben el nombre de verbos, que designan las acciones a realizar (por ejemplo, LIST, SORT...). En una orden se incluye el verbo, el nombre del fichero sobre el que actúa el verbo, el criterio de selección y clasificación u ordenacion, y los atributos y modificaciones del informe que se obtiene como salida.

2. Un compilador de BASIC, modificado y adaptado para el desarrollo de aplicaciones de gestión. Tiene sentencias estructuradas, como CASE, IF... THEN... ELSE, FOR... NEXT, LOOP... WHILE, LOOP... UNTIL, FOR... WHILE y FOR... UNTIL, que facilitan mucho el trabajo con este lenguaje.

3. Un lenguaje, llamado PROC, para escribir y almacenar secuencias de órdenes que son utilizadas muy frecuentemente, y que pueden ser llamadas mediante una orden simple (algo parecido a los ficheros .BAT de MS/DOS).

4. Un intérprete de comandos, llamado TCL (Terminal Control Languaje). Entre las órdenes que reconoce se encuentran las de crear nuevas cuentas para los usuarios, modificar la fecha y la hora, enviar mensajes a otros usuarios, y todas las relativas a la gestión de ficheros (borrar, copiar...).

En relación con los ficheros, en el sistema operativo PICK hay tres tipos, aunque los usuarios sólo se relacionan con dos de ellos: los **ficheros de datos,** y los **ficheros diccionario,** en los cuales se guardan las relaciones existentes entre los datos.

Dentro de los ficheros diccionario hay tres tipos:

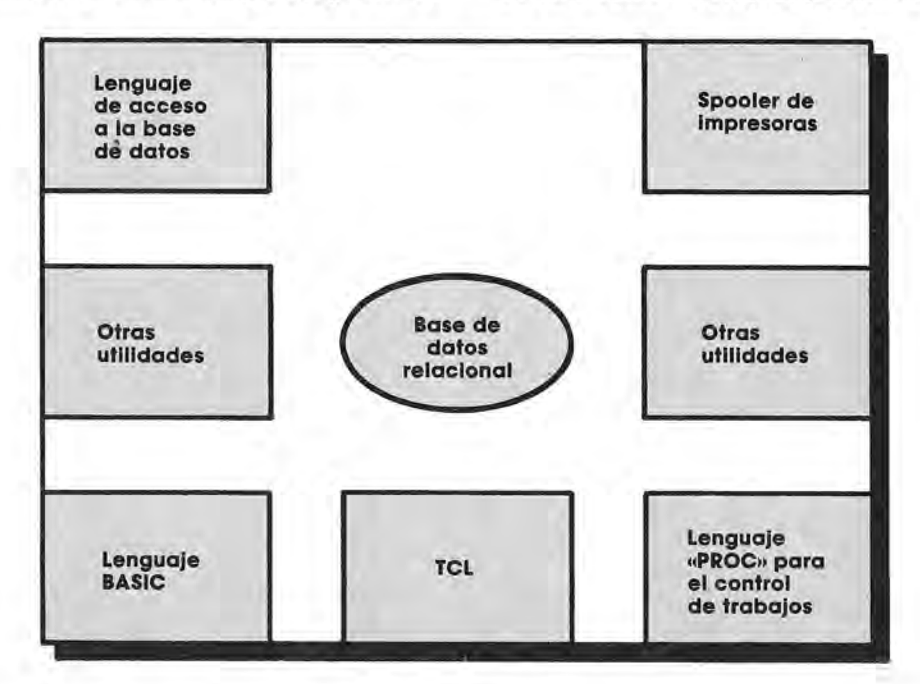
— El **diccionario del sistema,** que es donde están catalogados los usuarios.

— El **diccionario maestro.** Cada usuario tiene uno, en el cual se define el uso del sistema que puede hacer ese usuario.

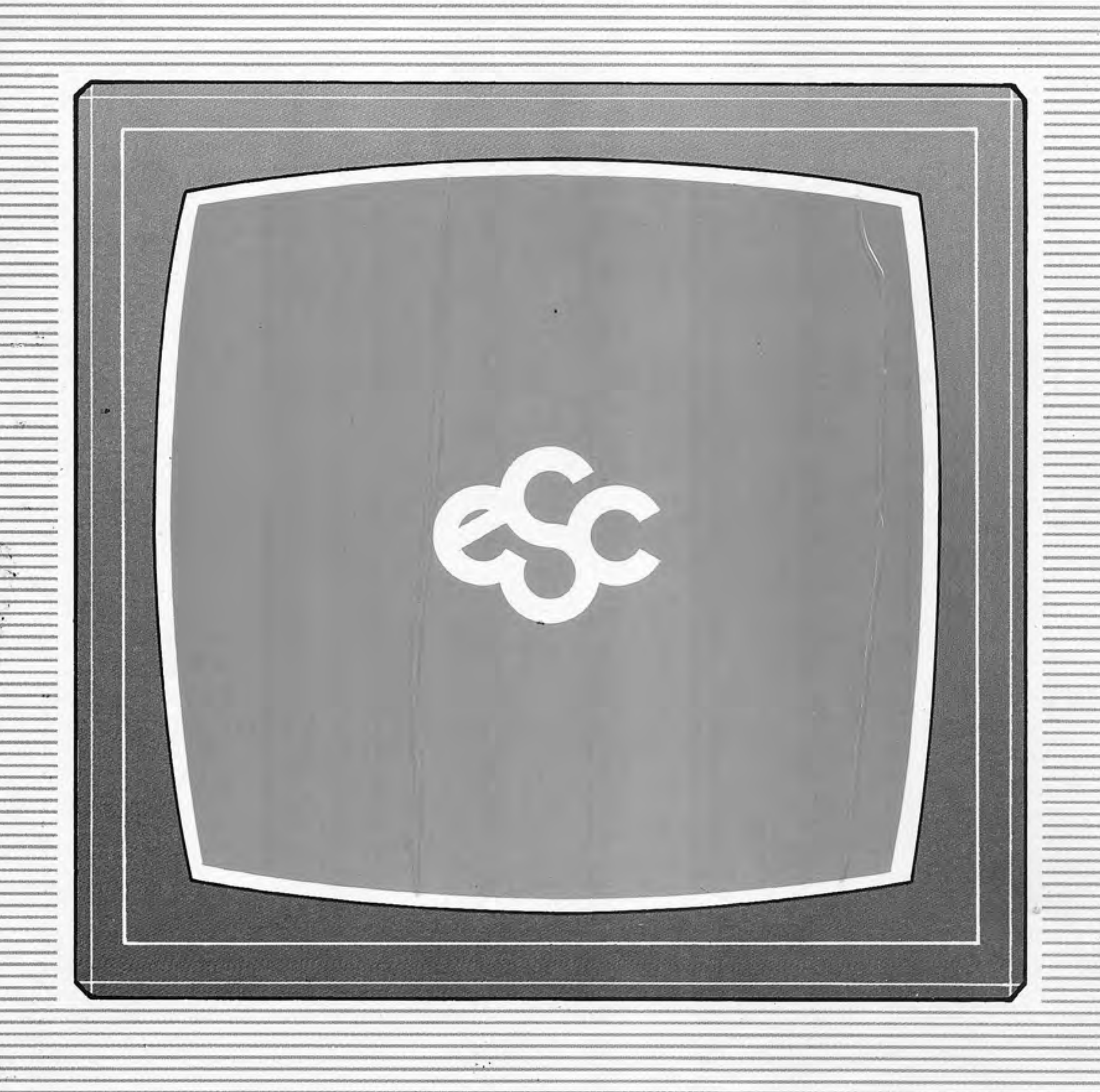
— El **diccionario de fichero.** A cada fichero de usuario se le asocia un diccionario de fichero en el cual se define su estructura general.

5. Un **«spooler»** de impresoras. Soporta varias colas de impresión, y se pueden controlar hasta 16 impresoras, no importa de qué tipo sean (de margarita, matriciales, láser...).

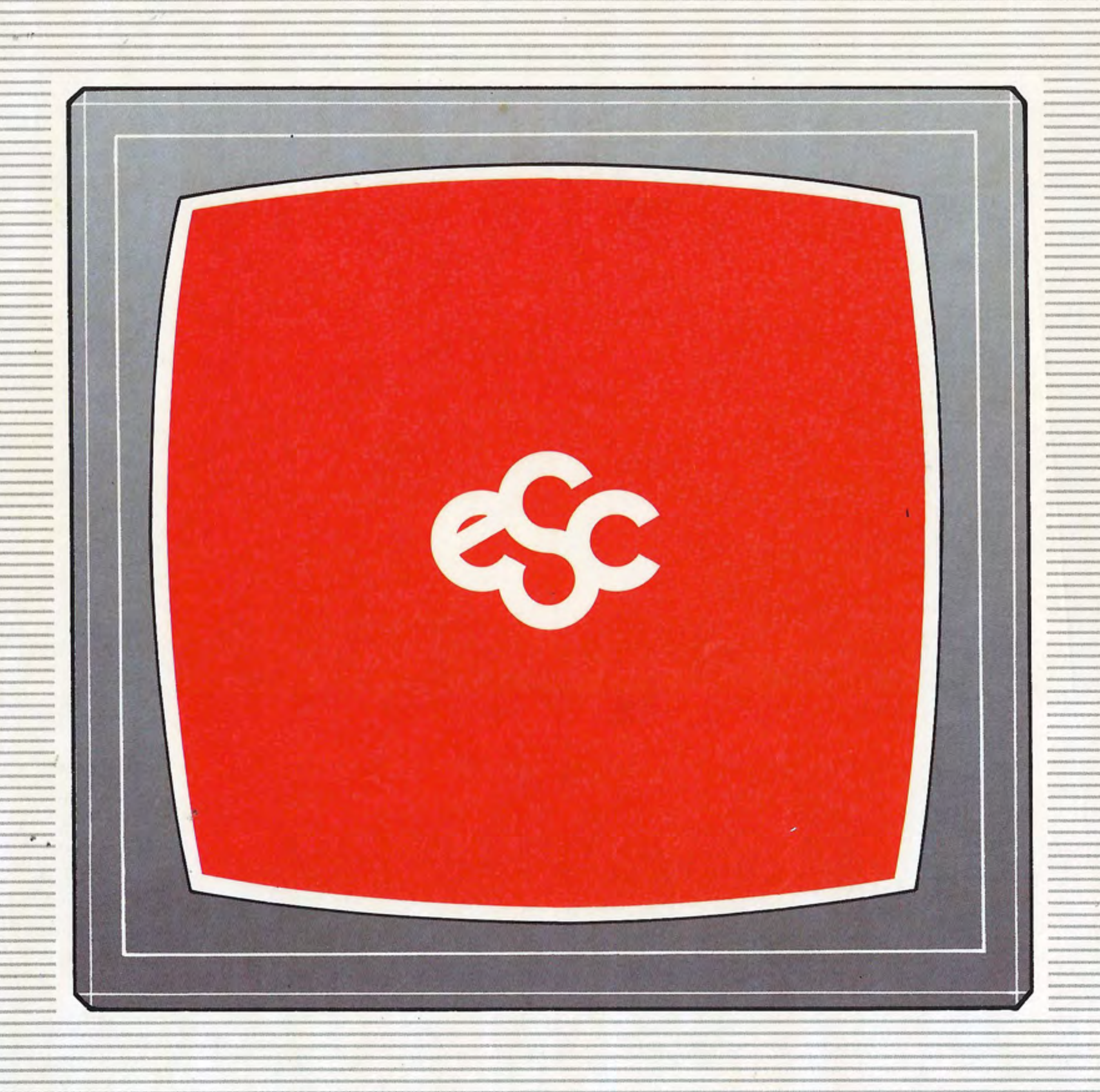
Además, PICK también dispone de otras utilidades encaminadas a facilitar el control de la información por parte de los usuarios.




Partes componentes del sistema operativo PICK.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼